Disseny i implementació d'un RDS basat en software lliure

Projecte final de Grau

Facultat d'Informàtica de Barcelona

Autor: Marc Filbà Gallego

Director: Xavier Martorell Bofill

Ponent: Òscar Linares Martín

Data: 19 de Gener de 2016

Índex

1. Introducció

1.1. Context

Aquest projecte és un TFG (Treball Final de Grau) en modalitat B, desenvolupat a l'empresa UPCnet. Bàsicament és un projecte que posarà en marxa un *RDS (Relational Database Service) [1]* basat en software lliure a la pròpia empresa.

Un RDS està format per un o un conjunt de servidors que ofereixen un servei de bases de dades. En aquest cas estarà format per un conjunt de servidors i per tant oferirà una sèrie d'avantatges sobre la implementació típica d'un servidor amb un *DBMS (Database Management System) [2]* com *MySQL [3]* o *PostgreSQL [4]*.

Entre aquests avantatges trobem la possibilitat d'un fàcil escalat horitzontal, per tal d'incrementar la productivitat del servei, alta disponibilitat, per assegurar la no interrupció del servei, i capacitat per recuperar les dades relatives a un determinat moment del temps, *PITR* (*Point In Time Recovery*) *[5]*, el qual és una assegurança sobre possibles desastres.

Actualment tenim desenes de servidor que tenen *DBMS's* instal·lats i que treballen independentment uns dels altres. Tot i que tots compleixen la seva funció a la perfecció, alguns d'ells no compten amb el servei d'alta disponibilitat i molts d'ells tampoc compten amb la opció del *PITR*. A més a més, el cost d'administració actual és molt més elevat que no el cost que dels sistema que es planteja. Per tant la intenció és que aquest nou sistema, molt més eficaç, segur i flexible acabi substituint els actuals servidors.

El model que s'implementarà consta de dues capes: *Front line* i *Second line*. La *Front line* està formada per dos nodes, però només un d'ells estarà actiu simultàniament, per poder assegurar l'alta disponibilitat. A la *Second line* hi trobem un node que farà de *Master* i dos nodes més que faran de *Slaves*. En cas de que hi hagués algun problema amb el *Master* que pogués comprometre el servei, el sistema ho detectarà i promocionarà un dels nodes *Slaves* a ser el nou *Master*.

Les dues capes treballen per resoldre cada una de les peticions i tenen el següent comportament: La primera capa és l'encarregada de rebre les connexions dels clients, tindrà un o més d'un punt d'entrada els qual serviran per accedir al servei. També haurà de catalogar la petició com a modificadora o consultora. La segona capa, rebrà la petició, l'executarà i retornarà el resultat a la primera capa, per tal que aquesta la faci arribar al client.

Si la petició que llença el client és del tipus modificadora, la *Front line* l'enviarà al *Master*, ja que és l'únic node que pot executar modificacions sobre la base de dades. Contràriament, si la petició no és modificadora, sinó consultora, la *Front line* l'enviarà a un dels tres nodes tot balancejant el nombre de peticions que resol cada un.

Per muntar aquest servei, tal i com s'explica i es detalla posteriorment en aquest mateix document, es comptarà amb algunes màquines virtuals i s'utilitzarà el *DBMS PostgreSQL* ja que actualment, la majoria de bases de dades estan muntades amb aquest gestor i seguint la llei d'Amdahl, millorant el que proporcionalment s'utilitza més, el guany és major.

En aquest projecte només s'inclourà suport per *PostgreSQL*, tot i que en un futur, és possible que s'acabi expandint l'RDS per oferir suport a *MySQL*. La versió de *PostgreSQL* que s'utilitzarà serà la 9.4 ja que, a part de ser la última que ha sortit, inclou una sèrie de millores per la construcció de *clústers* i pel mantenir la coherència dins d'aquests.

Per assegurar l'alta disponibilitat, es farà servir el programa *Replication Manager [6]*. Aquest programa actua com a *daemon* i monitoritza en tot moment el node *Master*. D'aquesta manera, tots els nodes saben quina és la posició dins el *clúster*, i qui és el *Master*.

Pel que fa a les còpies de seguretat, es valorarà si el mètode actual és viable amb aquest sistema, i si no ho fos, es buscarien alternatives que facilitin la recuperació de l'estat de les dades en un moment determinat del temps.

1.2. Actors implicats

En aquest projecte hi ha varies persones implicades, les podem classificar com directament i indirectament implicades. Pel que fa a les persones directament implicades trobem al desenvolupador del projecte, al director, Òscar Linares i a l'equip dins l'empresa. Dins el conjunt de les persones indirectament implicades en primer lloc trobem al professor Xavier Martorell, ponent d'aquest projecte el qual guiarà el projecte externament a l'empresa, assentant les bases i aconsellant en tot moment.

1.2.1. Desenvolupador

El desenvolupador no només s'encarregarà de la part tècnica, sinó que és el màxim responsable de tot el que implica el projecte. Implementació, proves, planificació, pressupost i assoliment dels objectius. També caldrà que generi tota la documentació necessària per tal que l'equip pugui administrar i gestionar el sistema.

1.2.2. Director

El director serà l'encarregat de guiar el projecte internament, però sobretot és qui valorarà si els objectius són els adequats i si s'ajusten a les necessitats de l'empresa. Possiblement hi ha haurà objectius que depenguin d'altres equips i ell és qui ajudarà a coordinar els diferents equips implicats.

1.2.3. Equip intern

Tot i que l'equip no col·laborarà activament en el desenvolupament, sí que són actors implicats directament ja que hauran de donar un suport molt important dia a dia al projecte i quan el projecte finalitzi, ells seran els que hauran d'administrar, gestionar i fer créixer el resultat.

1.2.4. Beneficiari

De beneficiaris, també podem trobar-ne de directes i d'indirectes:

- Directes: Com que el projecte es desenvoluparà a UPCnet, el màxim beneficiari és l'empresa, ja que gràcies a tenir un servei intern amb aquestes prestacions, els departaments interns podran treballar més còmodament. A més, podran oferir un servei als clients amb més avantatges respecte el que tenen ara, i amb un cost segurament inferior.

- Indirectes: Com a beneficiaris indirectes, trobem els clients finals que puguin consumir el servei, ja que com he comentat anteriorment, el servei que consumeixin tindrà avantatges respecte l'actual.

1.3. Estat de l'art

1.3.1. Definició de RDS

El concepte RDS no és gaire conegut ja que s'extreu del servei *AWS RDS [7]* ofert per l'empresa *Amazon* des de finals de l'any 2009.

Amb la revolució del *cloud computing [8],* l'any 2002 *Amazon* va començar a desenvolupar un conjunt de serveis orientats a desenvolupadors que va anomenar *AWS* (*Amazon Web Services*) *[9]*. Aquest servei el va llençar al mercat el 2006. Bàsicament oferia màquines virtuals totalment preparades on poder construir aplicacions sofisticades i escalables amb un seguit d'avantatges com poden ser el cost i la seguretat.

Amb el pas del temps es van trobar, que una gran quantitat de clients necessitaven un *DBMS* per guardar dades i que en una gran part de les màquines virtuals, hi havia una instància independent d'aquest *DBMS*. Així que el pròxim pas lògic, com a idea de negoci, va ser crear un

servei el qual implementés un *DBMS* global, on els usuaris podrien guardar les seves dades sense haver de tenir el *DBMS* instal·lat a la màquina on s'executava l'aplicació. El resultat d'aquest servei va ser l'*AWS RDS.*

Aquest nou servei va néixer només amb suport per a *MySQL* tot i que paulatinament es va afegir suport amb *Oracle Database*, *Microsoft SQL* i *PostgreSQL*. Actualment hi ha diverses iniciatives que ofereixin el servei *RDS* com per exemple *AWS RDS* del que hem parlat i *Google Cloud Platform [10]*.

1.3.2. Evolució de PostgreSQL

*PostgreSQ*L va tenir uns inicis una mica complicats ja que van apostar per un llenguatge que no era *SQL* i quan aquest es va estandarditzar, van haver de migrar el motor de la base de dades perquè treballés amb aquest llenguatge. A més a més, quan van acabar la migració, van canviar el nom del programa i el van anomenar *Postgre95*, cosa que no va afavorir-los gens. Finalment van canviar-li el nom altre cop, i el van nombre com el coneixem avui en dia, *PostgreSQL.*

Pel que fa a les versions, aquí tenim un breu resum de les funcionalitats més significatives:

La versió 7.1.3 va introduir l'opció de guardar cada operació modificadora que es feia sobre les dades en uns fitxers que es va designar amb el nom *WAL* (*Write-Ahead Logging*) *[11]*.

L'objectiu era guardar l'operació abans d'executar-la. En aquell moment segurament no tenia molt sentit, però per les versions posteriors, aquesta va ser una decisió clau.

Amb la versió 8.0.26 de *PostgreSQL* es va implementar una nova funcionalitat que permetia recuperar l'estat de les dades en un determinat moment del temps. Aquesta funcionalitat s'anomena *PITR* i està basada única i exclusivament en tractar el fitxer *WAL* de forma apropiada per poder recuperar extreure'n les operacions executades i reconstruir la base de dades en un moment concret del temps. A partir d'una còpia de seguretat de l'instant zero, es pot dur a terme aquesta recuperació, ja que només cal aplicar els canvis fins a la data i hora que es vol recuperar.

Però no va ser fins la versió 9.0.22 de *PostgreSQL* quan es van introduir una funcionalitat que permetia crear els primers *clústers* sense dependre de softwares de tercers, aquesta nova funcionalitats es va anomenar *Streming Replication [12].* A l'hora de crear un *clúster* amb aquesta nova funcionalitat, cal designar un node com a *Master* i un conjunt de nodes com a *Slaves*. El node *Master* és l'únic que té permís d'escriptura a les dades, i quan ho fa, sincronitza

el seu fitxer *WAL* amb els nodes *Slaves*. D'aquesta manera tots els nodes tenen la mateixa informació en temps real.

1.3.3. 2nd Quadrant

2nd Quadrant és una autoritat mundial en *PostgreSQL*. Ha fet significatives aportacions a aquest projecte, entre elles el *PITR* i l'*Streaming Replication*. A més a més ha desenvolupat diverses utilitats per la gestió d'un sistema *PostgreSQL*, un parell d'elles es faran servir en aquest projecte.

El més gran problema d'aquesta estructura *Master – Slave*, és que si cau el *Master*, el sistema es queda sense escriptures fins que manualment se solucioni el problema amb el *Master*, o manualment es canviï el rol d'un *Slave* per tal que es converteixi en *Master*.

Aquest problema és el que soluciona l'aplicació *Replication Manager*. Si hi ha algun problema amb el *Master*, que impedeix que desenvolupi aquest rol, aquesta utilitat és capaç de detectar-ho i si s'escau desautoritzar l'antic *Master* i convertir un *Slave* en nou *Master* .

L'altre utilitat es diu *Barman [13]* i es farà servir per facilitar la tasca de fer còpies de seguretat o restaurar-les, tot utilitzant PITR. *PostgreSQL* té una API completa per poder fer còpies de seguretat i també té mecanismes per fer una recuperació *PITR*. Però aquesta utilitat manté totes les còpies centralitzades i ofereix una interfície per fer i restaurar còpies de seguretat molt intuïtiva.

2. Abast del projecte

2.1.Formulació del problema

Aquest projecte neix amb la necessitat de tenir una plataforma única que ofereixi el servei de bases de dades. L'empresa cada cop té més servidors amb bases de dades i això es tradueix amb cost de manteniment i d'administració (tant econòmic com temporal).

Entre tots els servidors, veiem instal·lades 7 versions diferents de *PostgreSQL,* el quals entre elles no són estrictament compatibles. Per tant, a l'hora d'administrar, tampoc s'administren de la mateixa manera i a més a més, si un servidor d'una versió antiga falla, es pot donar una situació problemàtica

2.2.Objectiu

L'objectiu principal del projecte realitzat a UPCnet és preparar un sistema que acabi substituint les desenes de servidors actuals que tenen un *PostgreSQL* instal·lat. Cal dir que aquest projecte només muntarà el sistema inicial de manera que aquest sigui estable, segur i ampliable per la pròpia empresa, incloent la migració de les dades que ara hi ha en servidors de desenvolupament. La raó és que la migració de tots els servidors un cop el sistema estigui muntat, pot portar meso. I aquest és un temps del que no disposem en aquest TFG.

Així doncs, un cop definit l'objectiu principal, apuntem un seguit de sub-objectius els quals faran possible l'objectiu principal:

- Tenir l'*Streaming Replication* format pel node *Master* i els dos nodes *Slaves* muntat i funcionant, ja que aquesta sincronització és la base de tot el sistema.

- Gestionar els *failovers [14]* de manera que si el node Master cau el servei no es pengi, sinó que un node *Slave* agafi el rol del node caigut i es restableixi el servei.

- Valorar si l'opció que s'està fent servir ara per fer les còpies de seguretat s'adapta al nou sistema. És molt probable que no s'hi adapti ja que hi haurà menys servidors amb un volum de dades molt més gran.

- Aprofitar que tenim tres servidors que tenen les dades (ja que les dades estaran replicades en tots els nodes del *clúster*), per balancejar les peticions i per tant, obtenir un increment de rendiment sobre la implementació convencional.

- Facilitar l'administració i preparar documentació per tots els tècnics que posteriorment hagin d'administrar el sistema.

2.3. Requeriments

- El nou sistema ha de ser com a mínim igual de ràpid que l'actual.
- L'administració del sistema no pot ser molt més difícil que l'administració d'un servidor *PostgreSQL* típic.
- El sistema ha d'adaptar-se a l'entorn actual de l'empresa i no caldrà re-implementar aplicacions per assegurar el bon funcionament del conjunt.

2.4. Abast

El resultat d'aquest projecte serà un sistema que ofereixi un servei de bases de dades a la pròpia empresa.

Aquest servei, estarà implementat per una *Front line*, la qual s'encarregarà de balancejar les peticions, i d'una S*econd line*, la qual estarà formada per un node *Master* i dos nodes *Slaves* que serviran les peticions. Si la petició és modificadora, la servirà el *Master*, sinó no ho és, la servirà qualsevol dels tres nodes.

A més a més, al sistema resultant, s'hi haurà migrat algunes bases de dades que ara hi ha en servidors de desenvolupament.

2.5. Riscos i possibles solucions

Durant el projecte es poden donar algunes situacions problemàtiques. Per cada una d'aquestes s'intentarà donar solució apropiada:

2.5.1. No hi ha eines al nostre abast per desenvolupar el projecte

Fa temps, la mateixa empresa, va fer un estudi de viabilitat sobre aquest projecte i es va arribar a la conclusió que no hi havia eines a l'abast de l'empresa per tirar endavant el projecte.

Pot ser que durant l'estudi de viabilitat es torni a veure que no hi ha eines a l'abast de l'empresa per muntar aquest sistema.

Solució: En aquest cas, el projecte es centraria en construir un sistema basat en *MySQL* ja que se sap que existeixen eines que l'empresa podria fer servir per muntar el sistema amb aquest *DBMS*.

2.5.2. Burocràcia interna

Internament hi ha molta burocràcia a l'hora de crear màquines virtuals i preparar nous entorns. Possiblement aquesta burocràcia alenteixi el projecte i pugui acabar essent un problema.

Solució: Temporalment es poden crear les màquines virtuals a l'ordinador local. D'aquesta manera es podrien fer les primeres tasques sense tenir les màquines virtuals a l'entorn de proves de l'empresa. Tot i que per intentar solucionar aquest problema el més ràpid possible, cal mantenir una comunicació molt fluida amb el director i que si aquest problema sorgeix ell n'estigui al corrent de seguida.

2.5.3. Màquines de prova

Pot ser que les màquines virtuals no es creïn de bon principi a l'entorn definitiu sinó a un entorn de proves i que un cop allà no es puguin migrar a l'entorn definitiu, la qual cosa, podria causar retards en la planificació.

Solució: Cal comentar-ho amb el director i amb el responsable de infraestructures abans de la creació de màquines per tal d'intentar que aquesta situació no es doni ja que podria ser un problema greu.

2.6.Identificació de lleis i regulacions

Pel que fa a les lleis i regulacions, aquest projecte no n'afegeix respecte el sistema actual, ja que el projecte es basa en construir un sistema amb més possibilitats que l'actual i s'enfoca a poder tenir una millor gestió de les bases de dades. Per tant, les lleis del tipus LOPD *[15]* no depenen del sistema que gestiona les dades, sinó de com s'administren, i l'administració de les dades no canvia respecte al sistema actual.

2.7. Mètodes de treball

2.7.1. Metodologia Kanban

Durant aquest projecte es farà servir a metodologia *Kanban [16]* ja que ha semblat la millor opció tant pel director com pel desenvolupador. Algunes de les tasques que caldrà fer dependran d'altres equips i per tant, és una forma de tenir una visió global de l'estat del projecte.

Els avantatges d'aquesta metodologia és que es treballa amb per objectius i aquests estan escrits en targetes. Cada targeta té 3 estats, "Per fer", "fent" i "fet". Així doncs amb un cop d'ull es pot saber en quin estat està cada tasca.

2.7.2. Desenvolupament presencial

El projecte es durà a terme a l'oficina, ja que les màquines estaran en un entorn de prova que només és accessible des de les instal·lacions de l'empresa.

2.7.3. Reunions de seguiment

S'aniran fent reunions de seguiment periòdiques amb el director amb la finalitat de comprovar l'evolució del projecte. En aquestes reunions s'exposaran tant els avanços com les possibles traves que puguin sorgir.

2.7.4. Posada en marxa del sistema

Inicialment el sistema que pretén muntar aquest projecte es pot considerar com a independent ja que en primera instància no repercutirà sobre altres sistemes ja en funcionament.

Un cop el sistema sigui consistent passa a ser una infraestructura depenent ja que es començaran a migrar bases de dades d'aplicacions que ara hi ha en desenvolupament. A partir d'aquell moment el projecte es pot donar per pràcticament acabat ja que l'únic que faltarà és migrar les bases de dades i si escau, redimensionar el sistema. Cosa que permet gràcies al possible escalat horitzontal.

2.8. Eines de seguiment

Pel que fa al seguiment del projecte i seguint amb la metodologia *Kanban* s'utilitzarà una eina virtual on representar les targetes anomenada *Trello [17]*.

Pel que fa a les configuracions que s'apliquin i els *scripts* que es creïn, es desenvoluparan directament a les màquines de proves així que no es farà servir cap programa de control de versions.

2.9. Mètode d'avaluació

El sistema constarà de diferents parts, aquestes parts seran avaluables una a una, així que en cada fase del projecte es podrà validar la part que s'estigui implementant:

- Fase 1: Construir el clúster *Master - Slave*

Per validar el *clúster*, es crearà una base de dades en el node *Master* i aquesta s'ha de replicar al node *Slave*. Per contrari, si s'intenta crear una base de dades al node *Slave*, ha de sortir un error ja que aquest node no pot efectuar modificacions.

- Fase 2: Configurar la resolució de *failovers* automàtics

La segona fase es podrà validar parant el node *Master*. En el temps que indiquin les configuracions, un altre node *Slave* s'ha d'haver convertit en *Master*.

- Fase 3: Configurar el gestor de còpies de seguretat

Pel que fa a la tercera fase, caldrà fer una còpia de seguretat, fer una sèrie de modificacions a la base de dades, i recuperar l'estat de les dades abans de cada modificació.

- Fase 4: Configurar la capa balancejadora

Les proves que es faran servir per validar que la capa balancejador funciona correctament serà mitjançant proves de rendiment del sistema.

- Fase 5: Automatitzar procediments periòdics i facilitar l'administració el millor que es pugui (creant els *scripts* necessaris).

Quan el projecte acabi l'equip haurà d'administrar el sistema, per tant caldrà deixar-ho tot preparat per una gestió i administració el més còmode possible. Per validar aquest últim punt, caldrà comprovar que l'execució manual és la mateixa que la dels *scripts*.

3. Planificació

Aquest projecte el podem dividir en diferents tasques que es desglossen i s'expliquen en detall a continuació.

3.1. Estudi de mercat

La primera tasca és fer un estudi de mercat per comprovar que les solucions que es poden implementar amb *PostgreSQ*L compleixen els requisits i s'adapten tant al sistema actual com als recursos que l'empresa pot destinar al projecte.

L'estudi que s'ha fet ha inclòs des de solucions senzilles basades en l'emmagatzematge *NAS* (*Network-attached storage) [18]* fins a solucions molt més complexes que implementen sistemes distribuïts amb possibilitat de més d'un node *Master* per tal d'escalar l'escriptura horitzontalment. Per implementar la *Front line* s'ha optat per fer servir balancejadors *F5* els quals l'empresa ja disposa. Això si, la versió requerida per poder fer el balanceig és superior a la instal·lada i per tant caldrà aplicar una actualització per tal de poder-los fer servir per el projecte.

Per la part de la *Second line* s'ha valorat molt positivament l'evolució que està fent *PostgreSQL* pel que fa a la possibilitat de crear *clústers* a partir de mecanismes que aquest ja implementa. En les últimes versions han sortit moltes millores i afegint que a la conferència del 27 de Març del 2015 *PostgreSQL [19]* va anunciar que a les pròximes versions seria possible crear solucions *Multi-Master* nativament*,* s'ha cregut que no val la pena fer servir implementacions de tercers per fer el que *PostgreSQL* ja pot fer nativament.

Així doncs la solució proposada és crear un *clúster* a partir de la pròpia configuració de *PostgreSQL* i un parell de balancejadors *F5* (per tal d'assegurar l'alta disponibilitat). I pel que fa a la gestió del *clúster* es faran servir les eines *repmgr* i *barman* dels desenvolupadors *2nd Quadrant [20],* explicades anteriorment*.*

Feature	Shared Disk Failover	File System Replication	Transaction Log Shipping	Trigger-Based Master-Standby Replication	Statement-Based Replication Middleware	Asynchronous Multimaster Replication	Synchronous Multimaster Replication
Most Common Implementation	NAS	DRBD *[21]*	Streaming Repl.	Slony *[22]*	pgpool-II *[23]*	Bucardo *[24]*	(Pròximament)
Communication Method	shared disk	disk blocks	WAL	table rows	SQL	table rows	table rows and row locks
No special hardware required		•	•	•	•	•	•
Allows multiple master servers					•	•	•
No master server overhead	•		•		•		
No waiting for multiple servers	•		with sync off	•		•	
Master failure will never lose data	•	•	with sync on		•		•
Standby accept read-only queries			with hot	•	•	•	•
Per-table granularity				•		•	•
No conflict resolution necessary	•	•	•	•			•

Taula 1: Diferents solucions presentades per PostgreSQL [25]

3.1.1. Solucions actuals presentades per PostgreSQL

Les solucions que actuals existeixen per implementar el sistema que es necessita, basant-nos de moment en la no pèrdua de dades, són les següents.

3.1.1.1. NAS

Aquesta solució es basa en tenir un disc en xarxa en el que es guardin les bases de dades i els servidors que formin el sistema tinguin accés a ell tant de lectura com d'escriptura.

Segons la seva definició podem veure que aquest tipus de compartició de dades pot causar problemes de corrupció si més d'un node intenta accedir a les dades simultàniament.

Per tant aquesta solució només podria utilitzar-se en el cas que hi hagués un sol node actiu i la resta estiguessin inactius.

3.1.1.2. DRBD

La solució implementada amb *DRBD* es basa en duplicar el disc (a nivell de disc) a cada servidor i d'aquesta manera poder evitar el cas de corrupció de dades anterior ja que es modifiquen discs diferents i el programa només ha de sincronitzar els canvis. L'estructura canvia una mica respecte *NAS* i apareix el concepte *Master – Slave*.

DRBD també és una solució factible i pràctica en sistemes actiu – passiu ja que els nodes passius tenen les mateixes dades que l'actiu i en cas de fallada un d'ells podria assumir el rol de node actiu.

El problema principal, ve descrit a la seva documentació i esdevé perquè *DRBD* no és capaç de detectar la pèrdua de consistència que es pugui causar a les dades ja que treballa a nivell de disc que és un nivell inferior. És a dir, si les dades d'un node es corrompen, tots els nodes es corrompran perquè *DRBD* no és capaç de detectar-ho.

3.1.1.3. Streaming Replication

L'*Streaming Replication* és una funcionalitat que apareix amb PostgreSQL 9.0 i es basa en el mateix que *DRBD*, estructura *Master – Slave* i que cada node tingui una còpia local de la totalitat de les dades. Aquest sistema també permet un creixement horitzontal (per les lectures) per les modificacions no ho permet ja que només permet un node *Master*.

La diferència rau en que no ho fa a nivell de disc ni a nivell de fitxer, sinó a nivell de transacció. És a dir, partint d'un moment en que dues màquines tenen les mateixes dades, les modificacions que es fan en el node *Master* es repliquen al node *Slave*. 16

Aquest sistema es controla des de *PostgreSQL*, i si que hi ha control de corrupció de dades de manera que si un disc es corromp el mateix *PostgreSQL* donarà error i no propagarà la corrupció.

3.1.1.4. Pròximament

Segons l'última conferència de *PostgreSQL*, les funcionalitats especificades en aquesta columna seran natives en les properes versions de *PostgreSQL*, però de moment no hi ha cap sistema (recomanat per *PostgreSQL*) que les ofereixi.

3.2. Muntatge del sistema

3.2.1. Construir el clúster Master – Slave

Per configurar el *clúster*, es crearan 3 màquines en un entorn de proves el qual després permetrà una migració a l'entorn definitiu.

Les màquines es crearan a partir d'una plantilla pre-configurada per l'empresa la qual instal·larà *Ubuntu 14.04,* ja que és la última versió *LTS* (*Long Term Support) [26]* i ve configurada amb la monitorització, el programari de desplegament de canvis i amb el gestor de còpies de seguretat (a nivell de discs lògics).

Les màquines tindran dues targetes de xarxa. Una d'elles per donar el servei i l'altre estarà connectada a una xarxa privada la qual faran servir per tota la comunicació interna.

Sobre aquestes tres màquines s'instal·larà la versió 9.4 de *PostgreSQL* i es configurarà el fitxer *postgresql.conf*, habilitant l'*Streaming Replication*, i el fitxer *pg_hba.conf* el qual actual actua com a *ACL* (*Access Control List) [27]* i per tant cada màquina podrà accedir als *PostgreSQL* de les altres dues de forma segura i sense demanar contrasenya. També caldrà configurar el primer node inicialment com a *Master* i els altres dos com a *Slaves.*

Aquest punt només garanteix que els tres servidors tindran les dades, i que un canvi al servidor *Master* es replicarà als altres nodes. El node *Master* no donarà la petició per executada fins que el canvi no sigui efectiu als altres nodes *Slaves*.

Cal dir que a part de la configuració necessària per configurar l'*Streaming Replication*, també caldrà introduir la configuració necessària per tal de que tot funcioni correctament dins l'entorn de l'empresa.

3.2.2. Configurar la resolució automàtica dels failovers

Un cop tinguem el *clúster* bàsic muntat, s'haurà de configurar la resolució automàtica dels *failovers*. Per aquesta tasca s'instal·larà el programa *repmgr* i es configurarà a partir del fitxer *repmgr.conf* a cada una de les màquines. En ell, s'hi especificarà que si la connexió amb el node Master es perd durant 25 segons, un dels nodes *Slaves* agafi el rol de *Master*. El criteri que es farà servir per designar el nou *Master* serà, en aquest ordre: prioritat, a quants nodes del *clúster* té accés el propi node i número de node dins el *clúster*. Aquest programa crea una base de dades on emmagatzema les dades referents als nodes que formen el *clúster*.

A més a més, caldrà posar també tota la informació necessària perquè tant afegir nodes com recuperar nodes es pugui fer directament amb aquesta eina, per tal de facilitar l'administració.

També s'haurà de donar permís al programa modificant el fitxer *pg_hba.conf* de *PostgreSQL*. El primer servidor s'haurà configurat inicialment com a node *Master* i els altres dos com S*laves.*

Finalment s'haurà configurat per a que cada minut verifiqui que el *daemon* que gestiona els *failovers*, a partir del fitxers *repmgr.conf*, s'estigui executant, i si no s'està executant, que l'iniciï ja que sense aquest *daemon*, no es podran resoldre els *failover*s automàticament.

Després d'acabar aquest tasca, les màquines es podran migrar a l'entorn definitiu.

3.2.3. Configurar el gestor de còpies de seguretat

Pel que fa a les còpies de seguretat, s'ha analitzat el mètode actual i s'ha decidit que no es podrà implementar el mateix mètode en aquest sistema, ja que hi ha massa dades de les que fer còpia de seguretat.

Només un servidor caldrà que faci les còpies, ja que tots tres tindran les mateixes dades. Per aquesta tasca s'ha designat com a encarregat el tercer servidor al qual se li assignarà un disc extra, s'hi instal·larà el programa *barman* i es configurarà per tal que pugui accedir a la resta de servidors.

Caldrà modificar el fitxer *barman.conf* indicant la direcció i nom d'usuari entre altres dades, i als altres nodes s'haurà de tornar a modificar el fitxer *pg_hba.conf* per tal que el programa pugui accedir per poder fer la còpia de seguretat. Aquest programa actua com si fos un servidor independent i demana una rèplica de les dades al servidor *Master*, per tant, farà falta tocar tots els servidors per tal que permetin 3 connexions de replicació.

Aquest programa és una interfície que utilitza directament la *API* de *PostgreSQL* per fer les còpies de seguretat, però per l'usuari és molt més fàcil i còmode tant la creació de la còpia, com la gestió d'aquestes com la recuperació de les mateixes.

Pel que fa al rendiment durant les còpies de seguretat es creu que millorarà respecte el mètode anterior ja que aquest mètode no carrega tant el processador i no actua sobre el disc on accedeix *PostgreSQL*, sinó d'un disc diferent .

Al tercer servidor se li haurà de re-configurar el *repmgr.conf* per baixar-li la prioritat a l'hora resoldre els *failovers*. D'aquesta manera, aquest servidor només agafarà el rol de *Master* si no hi ha més nodes que ho puguin ser.

Per altra banda es crearà un *script* que executarà les comandes *barman* necessàries per fer la còpia de seguretat per tal d'automatitzar-ho.

Aquest tipus de còpia ens permet recuperar l'estat del conjunt de bases de dades en un moment determinat, però no ens permet recuperar una sola base de dades. Per recuperar les bases de dades per separat, caldrà crear un altre *script* que faci la còpia amb el mètode tradicional utilitzant *pg_dump [28],* això sí, només caldrà fer aquest tipus de còpia de seguretat en les taules més crítiques.

3.2.4. Configurar la capa balancejadora

Aquest pas serà dels més farragosos ja que els balancejadors *F5* que s'han de fer servir estan donant servei a altres entorns, de manera que s'haurà de fer un projecte intern a petita escala per tal de gestionar l'actualització a la nova versió. Un cop actualitzat s'haurà de configurar per tal que les operacions modificadores les enviï al node *Master* i les operacions consultores a un node *Slave.*

Aquests balancejadors, a partir de la versió 10.5 poden detectar si una petició és modificadora o no i aplicar polítiques en funció d'això. A més a més, també ofereixen un sistema per detectar quin dels nodes és el *Master* per tal de que si aquest canvia, els balancejadors se n'adonin. Aquest sistema ve donat pel fabricant en forma de plantilla. Només cal especificar quin *DBMS* es farà servir i automàticament es crearà un fitxer de configuració que amb un parell de modificacions podrem fer servir per aquesta tasca.

3.2.5. Automatitzar procediments periòdics i facilitar l'administració

Per tal de facilitar la gestió de tot el *clúster* es preveu que caldrà crear cap a una dotzena d'*scripts* per, entre d'altres coses:

- Iniciar/Aturar el servei, per a cada servidor
- Iniciar/Aturar el *daemon* gestor dels *failovers*, per a cada servidor
- Afegir/Eliminar nodes
- Inicialitzar el nou node amb tota la informació necessària per començar a funcionar
- Recuperar un node caigut

3.2.6. Documentació final

Com a última tasca, caldrà crear tota la documentació referent al projecte, no només com s'ha muntat i quines configuracions s'han fet servir i perquè, sinó també es redactaran procediments diversos que expliquin des de com estan fets els *scripts*, fins a com fer l'addició d'un nou node o com recuperar un node que abans era *Master.*

3.3. Migració de les bases de dades

Dins aquest TFG no hi ha prou temps per migrar totes les dades dels servidors de l'entorn de desenvolupament, per tant, un cop muntat el sistema se seleccionaran els servidors més antics d'aquest entorn i es migraran les bases de dades a aquest sistema.

3.4. Valoració d'alternatives i pla d'acció

A l'estudi de mercat es van presentar diferents propostes que s'adaptaven a l'entorn que s'està fent servir i a les característiques tant de les màquines i la xarxa, com dels serveis que ja estan funcionant i tenen contingut emmagatzemat en bases de dades *PostgreSQL*.

Un cop triada una de les propostes amb tots els seus avantatges i inconvenients serà difícil canviar algun aspecte, tot i així, s'han tingut en compte possibles alternatives sobre un parell de les tasques anteriorment explicades. La resta de tasques

3.4.1. Alternativa al canvi del mètode de còpia de seguretat

Se sap que el mètode de còpia de seguretat actual a llarg plaç no serà viable ja que la quantitat de dades a còpia serà exageradament gran comparada amb l'actual. Per aquest motiu, si hi ha problemes en complir la planificació, temporalment es deixaria el mètode actual. Tot i que com a TFG podria ser una solució viable, ja que es complirien tots els objectius, com a projecte intern caldria dedicar més temps i buscar una millor solució.

3.4.2. Alternativa a la utilització dels balancejadors

En aquesta tasca, depèn exclusivament de que un projecte intern s'iniciï i s'executi satisfactòriament. Aquest projecte intern no té data establerta d'execució així que l'alternativa és muntar un parell de servidors més que facin de *Front line*. Per a la implementació d'aquesta alternativa, es faria ús un programa anomenat *pgpool-II*. Aquest programa actua com a intermediari entre la petició i el *clúster* de *PostgreSQL,* també és capaç de catalogar les peticions i de balancejar-les, de manera que a la pràctica el seu comportament és el mateix que el dels balancejadors *F5*. Quan arribi el moment d'executar la tasca de muntar la *Front line* es valorarà si s'executa el pla inicial o si s'aplica l'alternativa.

La importància de les taques és vital pel projecte ja que sense una d'elles seria un projecte incomplert. Per realitzar aquest projecte cal executar les tasques amb suficient temps i cura per tal de deixar-ho tot lligat i funcionant. També és molt important anar documentant els passos fets ja que si es comença a documentar el procés quan el muntatge ha finalitzat ens serà impossible fer una documentació útil i de qualitat.

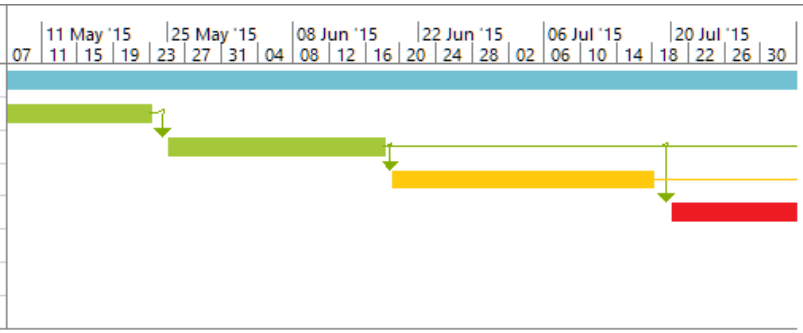
El projecte ha d'estar acabat a finals d'Octubre per tal de complir la planificació, poder afrontar possibles entrebancs i poder fer la presentació dins el període acordat.

Tot i la inexperiència de l'equip en aquest tema i la complexitat del projecte en si, s'espera acabar el projecte en els pròxims dies. Tot i això el resultat final del projecte és un sistema que caldrà mantenir i fer créixer per tal d'extreure tots els avantatges que pot aportar.

Es deixarà la part de gestió de les còpies de seguretat pel final i en el cas que es prevegi no poder complir la planificació, es faria l'alternativa i no el que està previst. Tot i que no s'implementés aquesta part dins el TFG, l'empresa dedicaria més recursos per tal que el projecte final si que ho implementés ja que com es comenta, a la llarga pot ser un problema significatiu.

3.5.Gantt planificació inicial

ID	Task Name	Duration	Start	Finish
1	Projecte (Dates inicial I final)	127 days	Thu 07/05/15	Fri 30/10/15
2	Estudi de mercat i tria de la proposta	12 days	Thu 07/05/15	Fri 22/05/15
3	Construir els Cúster Master - Slave	18 days	Mon 25/05/15	Wed 17/06/15
4	Resolució automàtica de failovers	21 days	Fri 19/06/15	Fri 17/07/15
5	Còpies de seguretat	25 days	Mon 20/07/15	Fri 21/08/15
6	Capa balancejadora	22 days	Mon 24/08/15	Tue 22/09/15
7	Automatitzar procediments periòdics I facilitar l'administració	18 days	Wed 23/09/15	Fri 16/10/15
8	Documentació final	10 days	Mon 19/10/15	Fri 30/10/15



ID	Task Name	Duration	Start	Finish
1	Projecte (Dates inicial I final)	127 days	Thu 07/05/15	Fri 30/10/15
2	Estudi de mercat i tria de la proposta	12 days	Thu 07/05/15	Fri 22/05/15
3	Construir els Cúster Master - Slave	18 days	Mon 25/05/15	Wed 17/06/15
4	Resolució automàtica de failovers	21 days	Fri 19/06/15	Fri 17/07/15
5	Còpies de seguretat	25 days	Mon 20/07/15	Fri 21/08/15
6	Capa balancejadora	22 days	Mon 24/08/15	Tue 22/09/15
7	Automatitzar procediments periòdics I facilitar l'administració	18 days	Wed 23/09/15	Fri 16/10/15
8	Documentació final	10 days	Mon 19/10/15	Fri 30/10/15

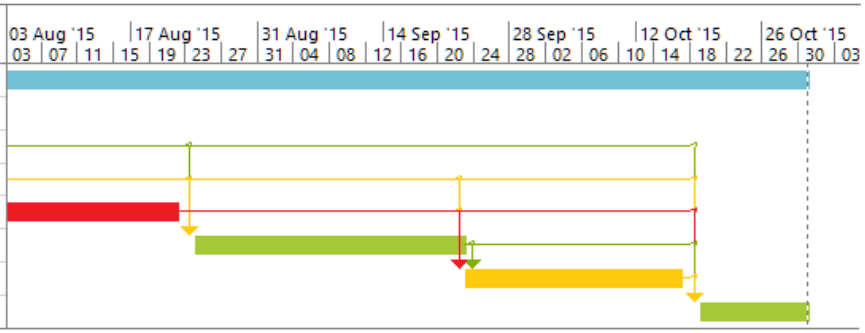


Figura 1: Gantt amb la planificació inicial.

En colors podem veure la criticitat de les tasques: vermell > groc > verd

4. Pressupost i sostenibilitat

4.1. Identificació i estimació de costos

En aquest projecte trobem diferents costos ben definits. Per una banda trobem el cost de la *First line* la qual s'ha implementat directament a l'entorn definitiu . El cost de la *Second line* a l'entorn de prova (cloud de la UPC), el cost de la d'aquest a l'entorn definitiu, la instal·lació i configuració i la importació de les diferents bases de dades de l'entorn desenvolupament.

A l'hora d'estudiar els costos, però, podem separar els costos per tasques, d'aquesta manera podem fer previsions i atendre a les possibles desviacions d'una forma més efectiva.

El preu hora del tècnic s'ha obtingut a partir del que, en mitja, es paga als tècnics externs.

Pel que fa el cost mensual dels entorns s'ha extret del que li costa a l'empresa una màquina a cadascun dels entorn. Els preus els té calculats l'empresa per tal de controlar els costos.

4.1.1. Cost de l'estudi de mercat

Per comprovar la viabilitat del projecte, s'ha tornat a fer l'estudi el qual ja es va fer a l'estiu del 2012 sense èxit ja que es va veure que muntar un sistema amb aquestes característiques no estava a l'abast de l'empresa. Aquest cop però, s'ha vist que l'empresa si que ho té a l'abast i per tant el projecte ha seguit endavant.

El cost de l'estudi ha estat de 40 hores i el preu de l'analista que l'ha dut a terme ha estat de 40€/hora, per tant l'estudi ha tingut un cost de **1600€.**

4.1.2. Cost del muntatge del sistema

Com hem vist a la definició de tasques, el muntatge es divideix en sis tasques, les quals seran analitzades a continuació.

4.1.2.1. Cost construcció del clúster Master - Slave

Aquesta tasca es desenvoluparà a l'entorn de proves, es composa de la suma de dos sub-costos i s'estima una duració de 18 dies. El cost total serà de **1845,95€** que es desglossen de la següent manera.

4.1.2.1.1. Cost del tècnic

El tècnic destinarà un total de 45 hores a la construcció del *clúster* i cada hora tindrà un cost de 40€. Per tant el cost del tècnic en executar aquesta tasca serà de 45hores*40€/hora = **1800€**

4.1.2.1.2. Cost de la infraestructura

Per a executar aquesta tasca es necessitaran tres màquines virtuals a l'entorn de prova, les quals tenen un preu de 25,53€/mes, per tant el cost de les 3 màquines en un mes serà de 3*25,53€/mes = 76,59€. Però dels 30 dies que te un mes, aquesta tasca en durarà 18, per tant el cost de la infraestructura per aquesta tasca és de 76,59*18/30 = **45,95€.** En el cas que la burocràcia alentís la creació de les màquines, aquestes es crearien en la màquina local i per tant el cost seria **0€.**

4.1.2.2. Cost de la Resolució automàtica de failovers

Aquesta tasca també es compon de dues sub-tasques. El temps estimat és de 21 dies i el cost total d'aquesta tasca és de **2253,61**€ que es desglossen de la següent manera.

4.1.2.2.1. Cost del tècnic

Tot i que es coneix el funcionament a nivell teòric del que es vol muntar, ni l'equip ni el tècnic tenen coneixements suficients per poder-ho implementar, així que s'estima que una part del temps es dedicarà a obtenir documentació per poder executar la tasca satisfactòriament. S'estima un cost de 55 hores a 40€/hora, per tant el cost del tècnic serà de **2200€**

4.1.2.2.2. Cost de la infraestructura

Les màquines seran les mateixes que es van crear en l'anterior tasca, per tant el cost serà de 76,59€*21/30 = **53,61€**. Igual que a l'anterior tasca, si la burocràcia alentís la creació de les màquines, aquestes es crearien en la màquina local i per tant el cost seria **0€.**

4.1.2.3. Cost de la configuració de les còpies de seguretat

Abans d'executar aquesta tasca, caldrà migrar les tres màquines virtuals a l'entorn definitiu, la migració és molt ràpida i fàcil de fer, s'estima que amb una hora es poden migrar les tres màquines. En comparació amb les hores destinades a aquesta tasca és un temps menyspreable, per tant comptarem que el temps destinat a aquesta tasca també inclou la migració. S'estima que es necessitaran 25 dies per poder executar aquesta tasca satisfactòriament. El cost total de la tasca serà de **4095,40€** els quals es desglossen de la següent manera.

4.1.2.3.1. Cost del tècnic

Aquest punt és el més crític ja que el problema més gran que ens podem trobar és que es perdin dades. Per aquest motiu és el punt on es dedicaran més hores. S'estima que es necessitaran

unes 100 hores per cobrir tant el fet de crear la còpia de seguretat com la posterior recuperació. El preu del tècnic serà doncs de: 100hores*40€/hora = **4000€**

4.1.2.3.2. Cost de la infraestructura

El cost mensual de cada màquina dins l'entorn definitiu és de 38,16€. Per tant el cost total d'aquestes màquines durant un mes seria de 3*38,16€ = 114,48€. Com que la tasca només durarà 25 dies, el cost de la tasca pel que fa a la infraestructura serà de 114,48*25/30 = **95,40€.**

4.1.2.4. Cost de la capa balancejadora

Com s'indica a l'apartat de definició de les tasques, aquesta tasca no té cost d'infraestructura ja que es compta amb dos balancejadors correctament configurats i que ja ofereixen un servei d'alta disponibilitat. El cost de la tasca doncs serà de **1200€**. En el cas que no es tinguessin aquests balancejadors disponibles, caldria muntar un parell de màquines virtuals i dedicar el temps del tècnic a la configuració d'aquesta nova *Front line* a més a més el cost de la tasca pujaria a **1255,97€**.

4.1.2.4.1. Cost del tècnic

Fer la còpia de seguretat de la configuració dels balancejadors, actualitzar-los i tornar a carregar la còpia de seguretat s'estima que costi unes 6 hores. La nova configuració i l'afinament dels paràmetres es creu que es podrà fer amb 24 hores més. Per tant el cost del tècnic per aquesta etapa serà de 30 hores*40€/hora = **1200€**

4.1.2.4.2. Cost de la infraestructura

En el cas que no es puguin fer servir els balancejadors F5, caldrà muntar dues màquines a l'entorn definitiu durant els 22 dies que durarà aquesta tasca. Per tant el cost serà de 2*38,16€= 76,32€ al cap del mes. La tasca però dura 22 dies, per tant el cost serà de 76,32*22/30 = **55,97€**.

4.1.2.1. Cost de l'automatització

Per facilitar l'administració es crearan diferents *scripts* i processos. S'estima que el procés d'automatització durarà uns 18 dies i tindrà un cost de **2068,69€**. Si la *Front line* no està formada pels balancejadors actuals sinó per les dues màquines extres, el cost puja a **2114,48€**.

4.1.2.1.1. Cost Tècnic

S'estima que el tècnic pot tardar unes 50 hores a fer tota la automatització, per tant el cost del tècnic en aquesta tasca serà de 50 hores*40€/hora = **1600€**.

4.1.2.1.2. Cost Infraestructura

Per tal de desenvolupar els *scripts*, es necessitaran les tres màquines funcionant, per tant el cost pel que fa a la infraestructura és de 114,48€*18/30 = **68,69€**

A més a més, si s'han hagut de crear la *Front line* a partir de dues màquines més i sense fer servir els balancejadors actuals, també cal comptar el seu cost que seria de 76,32*18/30 = **45,79€** extres**.**

4.1.2.2. Cost de la documentació

Finalment caldrà acabar de documentar els procediments a l'hora d'administrar el sistema. S'estima que tindrà una durada de 10 dies i que costarà uns **1638,16€**. Si la *Front line* no està formada pels balancejadors actuals, el cost s'incrementa a **1663,60€**.

4.1.2.2.1. Cost del tècnic

Pel que fa la documentació es creu que amb 40 hores serà suficient per enllestir-la. El cost del tècnic que faci la documentació serà doncs de 40 hores*40€/hora = **1600€.**

4.1.2.2.2. Cost de la infraestructura

Per a fer la documentació, és molt probable que el tècnic hagi de provocar situacions crítiques i verificar que aquesta ajuda a la resolució de les situacions, per tant caldrà que tot el sistema estigui funcionant. El cost doncs, serà de 114,48*10/30 = **38,16€**.

Com a les anteriors etapes, si la *Front line* no esta formada pels balancejadors actuals, el cost de les màquines extres puja a 76,32*10*30 = **25,44€**.

4.1.3. Migració de les bases de dades

Un cop creades i configurades les màquines a l'entorn definitiu, caldrà migrar 20 bases de dades d'altres servidors de desenvolupament, aquetes bases de dades s'hauran de compatibilitzar amb la nova versió de PostgreSQL, cosa que s'estima fer en 20 hores. El preu del tècnic també serà de 40€/hora, per tant el cost serà de 20 hores*40€/hora = **800€.**

4.2.Cost pressupostat

Tenint en compte totes les tasques anteriorment descrites, aquest és el cost total pressupostat per dur a terme aquest projecte.

Concepte	**Unitats**	**Preu**	**Cost total**
Estudi de mercat	40 hores	40€ / hora	1.600,00 €
Muntatge del sistema			
Construcció del clúster			
Tècnic	45 hores	40€ / hora	1.800,00 €
Infraestructura	3 màquines durant 18 dies	25,53€ / mes	45,95 €
Resolució *failovers*			
Tècnic	55 hores	40€ / hora	2.200,00 €
Infraestructura	3 màquines durant 21 dies	25,53€ / mes	53,61 €
Còpies de seguretat			
Tècnic	100 hores	40€ / hora	4.000,00 €
Infraestructura	3 màquines durant 25 dies	38,16€ / mes	95,40 €
Capa balancejadora			
Tècnic	30 hores	40€ / hora	1.200,00 €
Automatització			
Tècnic	50 hores	40€ / hora	2.000,00 €
Infraestructura	3 màquines durant 18 dies	38,16€ / mes	68,69 €
Documentació			
Tècnic	40 hores	40€ / hora	1.600,00 €
Infraestructura	3 màquines durant 10 dies	38,16€ / mes	38,16 €
Migració de les bases de dades	20 hores	40€ / hora	800,00 €
TOTAL			**15.501,81 €**

4.3.Desviacions

Cal tenir en compte que el pressupost no és el cost definitiu ja que per podria ser que la part balancejadora no es faci tal i com es planeja i per tant tindríem una desviació del cost. En aquest cas no seria gaire important, però podria donar-se'n alguna que si que ho fos.

En el cas que no es poguessin fer servir els balancejadors actuals, caldria afegir al pressupost la quantitat de 55,97€ + 45,79€ + 25,44€ = **127,20€.** Per tant tindríem una desviació ínfima de només el **0.8%**.

4.4.Sostenibilitat

4.4.1. Impacte econòmic

S'ha realitzat una avaluació de costos tenint en compte també les possibles desviacions. El cost real del projecte no és gaire elevat ja que la solució proposada s'implementarà dins l'empresa, amb recursos dels quals l'empresa ja disposa. A més a més, el fet de reduir la quantitat de màquines i eliminar la multi versió portarà una estalvi econòmic significatiu. Per aquests motius li posem un **8.5** com a nota.

4.4.2. Impacte social

Entre d'altres beneficiaris, trobem l'equip que administrarà el sistema, sobretot perquè amb una sola formació seran capaços d'administrar tot el sistema. Aprofitant la resolució automàtica de *failovers* serà possible resoldre més precisament les incidències ja que no caldrà parar el servei per a realitzar tasques que ara si que ho requereixen. Per aquests motius li posem un **8.0** com a nota.

4.4.3. Impacte ambiental

Pel que fa l'aspecte ambiental es creu que tindrà un impacte positiu ja que s'eliminaran desenes de servidors (tant físics com virtuals) i se substituiran per màquines que seran només virtuals.

A més a més, els servidors virtuals actuals, que només s'utilitzen per emmagatzemar dades no tenen un consum de processador gaire elevat però tot i així ocupen un lloc dins la virtualitzadora.

Amb el nou sistema, cada node tindrà un ús més intensiu de processador, però hi haurà moltes menys màquines, de manera que mediambientalment parlant, és una millora. Per aquests motius li posem un **9.5** com a nota.

5. Disseny

A part de que el sistema compleixi amb les especificacions, s'espera d'ell que sigui el màxim robust possible i que permeti una certa escalabilitat. Aquests requisits han estat clau a l'hora de fer el disseny de la solució ja que un cop es comenci a implementar cada modificació que es vulgui fer caldrà tornar a ser avaluada i per tant es podria no respectar la planificació.

Com s'ha comentat en la planificació, el sistema està format per dues capes. La primera capa, *Front line*, s'encarrega de rebre les peticions dels clients, enviar-les a la *Second line* i finalment recollir el resultat d'aquesta i retornar-lo al client. La segona capa o *Second line*, agafa la petició que li envia la primera capa, l'executa i li retorna el resultat. A continuació, a la *figura 2* podem veure més a fons el flux de dades.

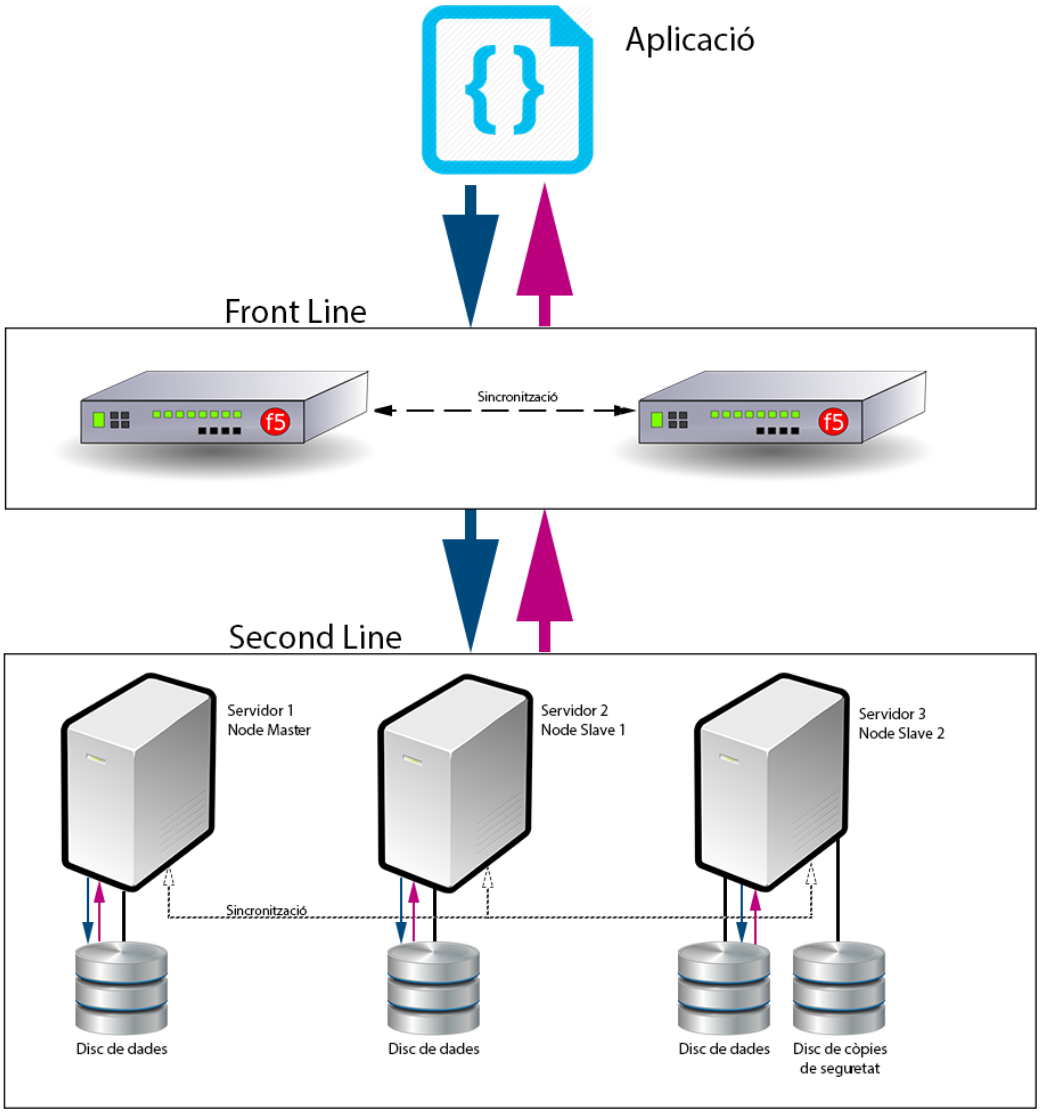


Figura 2: Disseny de les dues capes i representació del flux de dades.

5.1. Disseny de la Front line

La primera capa està formada per un parell de balancejadors F5 els quals ja es fan servir per a balancejar peticions *HTTP [29]* i *HTTPS [30]* en altres serveis que ofereix l'empresa.

Com que aquests balancejadors F5 ja estan donant servei, tota la part de sincronització entre els dos nodes ja està feta i per tant no la veurem en aquest projecte.

Segons l'estudi previ es va veure que calia actualitzar aquests balancejadors com a mínim a la versió 10.5, ja que aquesta és la mínima versió que es necessita per poder classificar les peticions en consultores i modificadores. Tot i això s'ha vist que l'última versió disponible és la 12.5.

Veient que hi ha algunes versions per sobre la versió 10.5, s'intentarà posar una versió més nova i actualitzada.

Per poder recollir les peticions, classificar-les i balancejar-les correctament, s'ha dividit aquesta primera capa en dues parts.

5.1.1. Resolució de les peticions

Per poder resoldre les peticions correctament, cal enviar-les on toca, si és modificadora, caldrà enviar-la al node *Master*, sinó ho és, cal balancejar-les als diferents nodes que formen la segona capa.

Per fer-ho es crearan dos *pools [31]* de servidors. Cada un dels *pools* contindrà els tres servidors però aquests podran estar actius o inactius.

L'objectiu doncs, és que un dels *pools* sigui per identificar quin dels nodes és el *Master*, i per tant, l'únic que pot executar escriptures. En aquest *pool* només hi haurà un node actiu simultàniament, el qual coincidirà amb el *Master* de la *Second line*.

El segon *pool* doncs, tindrà actius els nodes que puguin executar lectures (per defecte tots).

El mecanisme que tenen aquests balancejadors per detectar si un node està actiu o no, és executant una consulta a cadascun dels servidors.

Per la consulta que determina si un node és *Master*, s'ha analitzat la base de dades que manté *PostgreSQL* per poder fer l'*Streaming Replication* i s'ha dissenyar una consulta que retorna *True* si el node és el *Master* i *False* altrament.

Per identificar si un node pot fer lectures, és fa una petició simple a qualsevol base de dades ja que l'únic cas que no retornarà *True* serà perquè el node no està disponible.

5.1.2. Classificació i balanceig de les peticions

Per tal de poder classificar les peticions els balancejadors *F5* proveeixen d'un mecanisme anomenat *iRule [32]*. Aquest mecanisme no és més que un *script* que farà de *proxy [33]* per les peticions. Totes les peticions aniran a parar a aquest *proxy* i aquest haurà d'analitzar la petició i detectar si és modificadora o consultora.

En el primer cas, caldrà enviar-la directament a qualsevol node actiu del primer *pool.* A aquest *pool* només hi haurà un node actiu i per tant sempre aniran a parar al mateix servidor, al *Master*.

En el segon cas, la cosa es complica una mica ja que no podem enviar la petició a qualsevol node del segon *pool* sinó que cal balancejar-ho. Així doncs, caldrà disposar de la informació que ens indiqui a quin node cal enviar la petició.

Segons els manuals dels balancejadors *F5,* a partir de la versió 10.5 és possible utilitzar una plantilla pre-carregada pel fabricant la qual ja classifica les peticions. En aquesta plantilla caldrà ajustar el nom dels *pools* i fer la part del balanceig.

5.2.Disseny de la Second line

La segona capa, a diferència de la primera, s'haurà de crear des de l'inici i un cop creada, hi haurà aspectes que serà imprescindible que estiguin ben dissenyats, ja que sinó possiblement es perdi més temps intentant arreglar el problema que no pas implementant la resta de parts, així que cal tenir un bon disseny abans de començar amb la implementació.

Aquesta capa estarà formada per tres nodes, un d'ells serà el node *Master*, el que pot fer escriptures, i els altres dos només podran executar consultes.

El disseny d'aquesta capa també s'ha separat en diferents punts, els quals tractant-los per separat seran més fàcils de resoldre.

5.2.1. Característiques de les màquines

Les màquines inicialment tindran les mateixes característiques, però el sistema és prou flexible com perquè, si és necessari, les màquines puguin tenir característiques diferents, depenent d'on sigui el coll d'ampolla.

Com a punt positiu, les màquines seran virtuals de manera que sempre es podran assignar més recursos tant abans d'implementar la solució, com un cop implementada.

5.2.1.1. Característiques bàsiques

Cadascuna de les màquines tindrà 1 processador amb 2 cores i 1 GB. Donat que la majoria de servidors que tenim a l'empresa que tenen un *PostgreSQL* instal·lat tenen unes especificacions semblants, es creu que serà un bon punt de partida.

5.2.1.2. Característiques de disc

És imprescindible que cada una d'elles disposi d'un primer disc pel sistema operatiu i utilitats i un segon disc per emmagatzemar les dades. A més a més, cal que el segon disc, sigui una unitat lògica allotjada a la cabina de discs, per tal de complir amb els protocols de seguretat i *RAID [34]* del qual es disposa.

D'aquesta manera les dades estaran aïllades per si hi hagués un problema amb el disc principal i el rendiment del disc on hi ha les dades no es veurà afectat per la utilització del disc del sistema operatiu. Els discs es tractaran en format *LVM [35]* de manera que serà possible el redimensionament dels discs en calent, sense haver de reiniciar la màquina.

5.2.1.3. Característiques de xarxa

La comunicació entre els nodes és vital per a que aquesta capa funcioni correctament, per aquest motiu les màquines tindran dues targetes de xarxa. Una estarà connectada a una xarxa privada entre elles per tota la comunicació interna i l'altre és per on la *Front line* accedirà a cadascuna de les màquines.

5.2.1.4. Especificacions detallades

En aquesta taula podem veure el resum d'especificacions bàsiques, de disc i de xarxa que tindran cadascuna de les màquines.

Especificacions estàndards
1 Processador amb 2 cores a 3.4 GHz
1 GB de memòria RAM
2 Targetes de Xarxa
1 Disc de 20 GB per emmagatzemar el sistema operatiu
1 Disc de 20 GB per emmagatzemar les dades
1 Disc de 20 GB per emmagatzemar les còpies de seguretat (només 1 dels 3 servidors)

Taula 2: Especificacions de les màquines

5.2.2. Clúster Master-Slave

Per fer una instal·lació neta, s'inicialitzarà tota la base de dades, incloent els fitxers de configuració i els *scripts* que posteriorment caldrà fer per l'automatització de procediments, directament al segon disc, el que contindrà la base de dades.

La versió de *PostgreSQL* que es vol fer servir, encara no està als repositoris *d'Ubuntu*, per tant caldrà afegir el repositori de *PostgreSQL* 9.4 a cada una de les màquines.

La creació del *clúster* estarà totalment basada en l'*Streaming Replication* natiu de *PostgreSQL*. Aquest *clúster* resultant estarà format per un node que designarem com a *Master*, el qual podrà escriure i llegir la base de dades, i pels altres dos nodes que només podran llegir.

La idea general de l'*Streaming Replication*, és que el primer node que s'inicïi amb aquesta configuració habilitada crea una *timeline [36]*. Dins d'aquesta *timeline,* ell és el *Master*, però permet que altres nodes, que tenen la mateixa configuració s'hi associïn, de manera que el node *Master* comunica a tots els nodes associats les modificacions que va fent. Així les dades que tenen els nodes que formen el *clúster* estan sincronitzades.

Per altra banda també es configurarà el node *Master* perquè guardi els *WAL's* a una carpeta concreta, d'aquesta manera tindrem un històric de les modificacions que es van fent a la base de dades i per tant, podrem restaurar-la en cas de desastre.

L'*Streaming Replication* permet que un node s'associï a una *timeline* tant si el nou node té les mateixes dades que el node *Master* com si té una versió anterior de les dades. En aquest segon cas, el *Master* enviarà al nou node els *WAL's* necessaris per a què el nou node pugui actualitzar les seves dades. En cas que no hi hagi tots els *WAL's* necessaris, el nou node estigui a una *timeline* diferent o senzillament tingui unes bases de dades diferents, no podrà associar-se.

Un cop creat el *clúster*, tindrem un node *Master* i els altres dos nodes associats a ell, però no podrem detectar la caiguda d'un node.

5.2.3. Resolució de failovers

Aquesta part és de les més importats ja que dotarà el sistema de la capacitat de detectar la caiguda d'un node i d'actuar en conseqüència.

Per fer-ho s'utilitzarà l'eina *repmgr,* la qual tampoc està als repositoris d'*Ubuntu*, però sí que hi és als que ja hem afegit per poder instal·lar *PostgreSQL* 9.4. Aquesta eina requereix un fitxer de configuració el qual és diferent a cada un dels nodes. La informació que cal definir en aquest

fitxer és bàsicament la identificació d'aquest node dins el *clúster*, les dades necessàries per a que els altres nodes puguin accedir al seu *PostgreSQL*, quines comandes executar si el node *Master* cau i aquest node es converteix en nou *Master*, la mateixa situació anterior però que aquest node segueixi com *Slave* i finalment si el protocol s'ha d'executar automàticament o si volem fer-ho manualment.

Aquest programa actuarà com a *daemon* a cada un dels servidors i monitoritzarà l'estat del node *Master*, per tant caldrà que sempre s'estigui executant ja que si no ho fa, no podrem detectar les situacions de *failover*.

A més a més *repmgr* no té accés a la base de dades que *PostgreSQL* manté per *l'Streaming Replication,* així que a priori no sap quins nodes formen el *clúster*.

Així doncs, proveeix una sèrie de comandes les quals ens permetran crear una base de dades dedicada a *repmgr* per a què pugui monitoritzar el *clúster*. Com que només el node *Master* pot escriure, un cop tingui la base de dades, podrà identificar quin node és *Master* i quin és *Slave*.

El que més ens interessa d'això és detectar els *failovers* i que automàticament es resolguin. En cas de *failovers* el criteri que es fa servir per designar el nou *Master* és, en aquest ordre: prioritat, a quants nodes del *clúster* té accés, número de node. De manera que, tot i que els *failovers* es resolguin de forma automàtica, tenim control total a l'hora de decidir quin serà el nou *Master*.

5.2.4. Còpies de seguretat

Finalment arribem al pas més crític. El sistema construït anteriorment no té sentit si no podem afrontar situacions de desastre, pèrdua de dades, corrupció de discs...

Per afrontar aquest tema, caldrà fixar una sèrie de premisses. La primera de totes és que inicialment tindrem una quantitat de dades més aviat petita per poder fer proves còmodament. Al tenir tant poques dades, tenim un ample ventall de possibilitats per fer les còpies de seguretat, però això no sempre serà així. De manera que cal buscar mètodes enfocats a grans quantitats de dades.

PostgreSQL ofereix un parell d'eines per fer còpies de seguretat.

5.2.4.1. Còpies de seguretat amb pgdump

La primera, *pgdump* es basa en mirar les dades que conté la base de dades la qual es vol fer la còpia de seguretat i crear una llista de sentències SQL que quan s'executi generi la base de dades amb exactament el mateix contingut. Aquest mètode té els seus avantatges, com per exemple, que permet fer còpies de seguretat d'una taula dins una base de dades. Però també

inconvenients, com per exemple que tant la mida de la còpia com el temps necessari per fer-la és estratosfèrica.

5.2.4.2. Còpies de seguretat PITR

La segona, es basa en, partint d'una còpia de seguretat inicial, si es guarden els fitxers *WAL*, és possible restaurar la base de dades en un moment específic d'entre la còpia base i ara, aquest mètode de còpia de seguretat l'hem descrit abans, és l'anomenat Còpia de seguretat PITR. Aquest mètode té avantatges significatius respecte l'anterior mètode, però per contrapartida, una recuperació es fa de totes les bases de dades, de manera que si només vols recuperar una taula, és impossible.

La solució proposada doncs és implementar ambdós sistemes. El primer només es farà servir per taules puntuals, i el segon com a còpia del sistema.

L'encarregat de fer les còpies de seguretat s'ha decidit que serà el tercer node del *clúster*. Per a que les còpies no interfereixin amb el rendiment, s'ha triat l'últim node i se li ha baixat la prioritat, de manera que només es convertirà en *Master* si no hi ha més nodes al *clúster*.

Per poder fer les còpies de seguretat se li afegirà un altre disc, d'aquesta manera tot i que sigui *Master* el fet de fer la còpia de seguretat només consumirà *CPU* i *RAM*.

Com que l'empresa no acostuma a fer aquest tipus de còpia de seguretat, també s'instal·larà l'eina *barman,* la qual fa possible que amb molt poques comandes es pugui fer una còpia, revisar de quantes còpies es disposa, en definitiva, una bona gestió de les còpies de seguretat.

Per configurar *barman* correctament caldrà que el node *Master* no guardi els *WAL's* sinó que els enviï a aquest node, així *barman* els recollirà i els emmagatzemarà junt amb la còpia base. D'aquesta manera quan es vulgui recuperar una còpia de seguretat, *barman* podrà muntar l'estructura de directoris necessària perquè només especificant la data a la que es vol fer el *rollback [37]*, *PostgreSQL* sigui capaç de fer-ho.

El mètode que fa servir *barman* per fer la còpia de seguretat es basa en crear una connexió que se subscrigui a la *timeline* del node *Master.* A nivell conceptual és com si tinguéssim un altre node connectat al nostre *Streaming Replication,* la única diferència, és que un cop feta la còpia, aquesta connexió es trenca i no es torna a connectar fins la següent còpia de seguretat. L'esquematització d'aquest mètode el podem veure recreat a la *figura 3.*

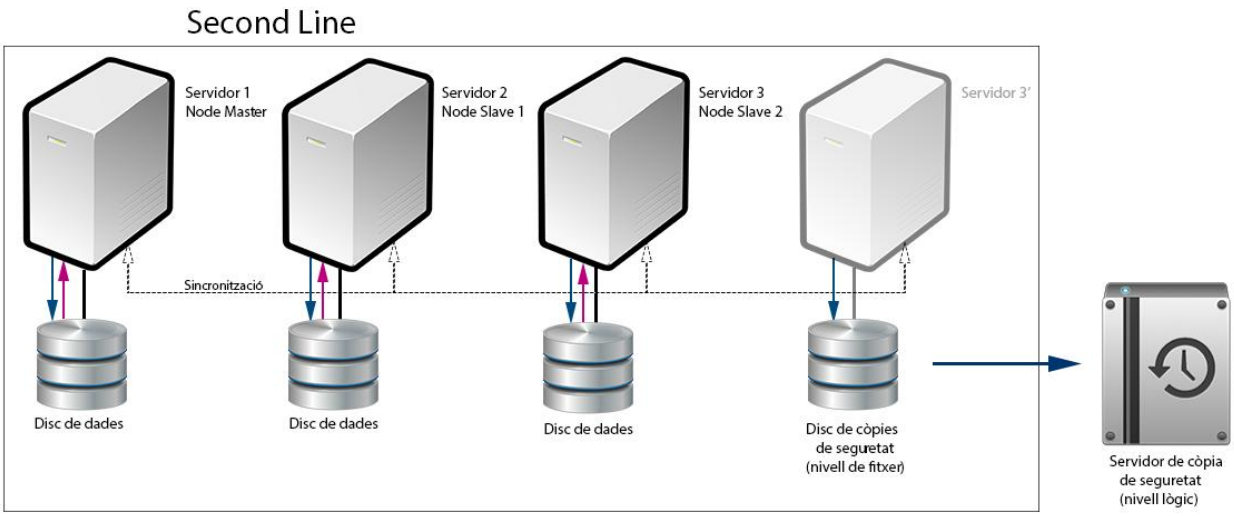


Figura 3: Representació de la Second Line a nivell conceptual.

Finalment, no ens interessa tenir una còpia de seguretat del sistema operatiu, només de les dades, i arribats a aquest punt, tenim 4 còpies de les dades. Una a cada node menys al tercer node, que tenim dues còpies de les dades. Amb aquesta estructura muntada, el segon nivell de còpia de seguretat, que es fa a nivell de disc lògic i no a nivell de fitxer, pot fer la còpia de seguretat del disc que hem afegit a al tercer node, i així en cap moment es malmetrà el rendiment del *clúster*.

6. Implementació

La implementació no seguirà el mateix ordre que s'ha definit a l'apartat de disseny ja que com s'ha comentat anteriorment, cal actualitzar els balancejadors que formen la *Front line,* i aquesta actualització s'executarà com a projecte a part dins el desenvolupament d'aquest.

Així doncs, s'ha començat per la *Second line* tot respectant l'ordre de tasques definit durant la planificació.

6.1. Creació de la Second line

6.1.1. Creació de les màquines virtuals

L'empresa disposa de dos entorns disponibles a l'hora de crear màquines virtuals, l'entorn de prova i l'entorn definitiu. Les tres màquines s'han creat de moment a l'entorn de proves com dicta el protocol intern. Cada màquina s'ha proveït d'1 processador amb 2 cores i 1 GB de memòria *RAM*. Se'ls han assignat també 2 targetes de xarxa, un disc dur de 20 GB on anirà a parar el sistema operatiu i un segon disc dur també de 20 GB ubicat a la cabina de discs, on s'emmagatzemaran les dades.

Acte seguit s'hi ha carregat la imatge d'*Ubuntu Linux 14.04 LTS* amb tot el programari necessari, tant els controladors com el programa de desplegament de canvis i el de còpies de seguretat a nivell de disc lògic.

Seguidament s'ha inicialitzat el disc on anirà a parar la base de dades amb format *LVM.* Aquest format permet crear una capa entre el disc físic (suport real) i el disc lògic (el que veu el sistema operatiu) de manera que el sistema operatiu vegi uns discs lògics creats a partir d'altres discs lògics en comptes del disc físic.

Els discs lògics que nosaltres crearem, poden estar formats per més d'un disc físic. D'aquesta manera podem redimensionar aquests discs lògics sense que el sistema se n'adoni i per tant sense haver de reiniciar-lo.

Creem els dics i especifiquem que volem que estiguin disponibles cada cop que iniciem la màquina.

6.1.2. Creació del clúster Master – Slave

Per la creació del *clúster Master – Slave* s'ha afegit el repositori de *PostgreSQL* 9.4 a les tres màquines i s'ha instal·lat el paquet. Acte seguit s'ha deshabilitat l'inici automàtic del servei quan s'engega la màquina ja que si tenim una màquina parada que anteriorment era *Master*, i

l'iniciem, iniciarà una altra *timeline* on ella continua essent *Master*, i tindrem 2 *Masters* treballant en *timelines* diferents, cosa que ens pot donar problemes de coherència de dades.

Per a que les màquines es puguin comunicar entre elles, hem generat claus RSA a cada una d'elles i les hem distribuït entre elles, de manera que l'usuari *postgres* de qualsevol màquina pugui accedir a qualsevol altra màquina.

Un cop fet això hem inicialitzat una nova base de dades al disc on guardarem tota la base de dades i hem aplicat la configuració necessària al fitxer *postgresql.conf* i al fitxer *pg_hba.conf.* Ambdós fitxes es troben a l'arrel de la carpeta on tenim la base de dades creada.

En el primer hem habilitat l'*Streaming Replication* i hem especificat que el *clúster* estarà format per tres nodes. També hem habilitat el *Continuous Archiving [38]* el quan anirà guardant els fitxers *WAL* com a històric de canvis fets a les bases de dades.

Finalment en el fitxer *pg_hba.conf* hem donat permisos a tots els membres del *clúster* per poder accedir a qualsevol altre node.

Tots els nodes del *clúster* tindran la mateixa configuració i les mateixes dades inicials, per tant el primer PostgreSQL que iniciem crearà la *timeline* (i per tant serà el *Master*) i a mesura que anem iniciant la resta de nodes s'aniran subscrivint a aquesta *timeline* i alhora es convertiran en *Slaves*.

6.1.3. Detecció i resolució automàtica de failovers

Tant per la detecció com per la resolució de *failovers* s'ha utilitzat l'eina *repmgr.* Aquesta eina necessita un fitxer de configuració on s'ha especificat la informació necessària:

- Nom del *clúster*
- Nombre de node
- Nom del node
- Dades referents a com accedir al *PostgreSQL*
- Dades referents a *timeout* i intents de reconnexió amb el servidor *Master*
- Comanda que cal executar si hi ha hagut un *failover* i aquest node s'ha de convertir en el nou *Master*
- Comanda que cal executar si hi ha hagut un *failover* i aquest node ha de continuar essent *Slave*
- Resolució dels *failover*s automàticament

Un cop especificada aquesta informació s'ha creat una base de dades i un usuari per a que aquest programa pugui funcionar correctament i finalment s'han afegit els nodes a la base de dades del *repmgr*.

Un cop arribats a aquí, la base de dades de *repmgr* conté la informació rellevant del *clúster*, així que cal iniciar-lo a cada node i assegurar-se que està executant-se en tot moment ja que si no fos així, la resolució automàtica de *failovers* es veuria compromesa.

El *repmgr* monitoritzarà el servidor *Master* a partir dels paràmetres de *timeout* i reconnexió que li hem especificat, en cas de que detecti que el node *Master* ha caigut, activarà automàticament el protocol de resolució de *failovers* i convertirà el node escollit en el nou node *Master*. El criteri per designar un nou *Master* és, en aquest ordre: prioritat, a quants nodes que formen el *clúster* puc accedir, i nombre de node.

Un cop acabada la implementació de la detecció i resolució automàtica dels *failover*s, s'ha valorat quina *Front line* s'implementaria, la inicial o l'alternativa ja que la inicial depèn d'uns recursos que s'estan fent servir i per tant cal que es comenci a moure per a que no hi hagi demores en la planificació. Es decideix que no cal implementar l'alternativa perquè des de *F5* hi ha documentació per implementar el que necessitem i per tant sembla la millor opció.

6.1.4. Implementació de les còpies de seguretat

Pel que fa a les còpies de seguretat, s'ha dividit en dues parts.

6.1.4.1. Còpia de seguretat del sistema operatiu

Habitualment no solem recuperar sistemes operatius sencers, per tant, fer còpia de seguretat periòdica d'aquests sistemes seria una pèrdua de temps, d'espai en disc i de rendiment a l'hora de servir les peticions.

Per aquest motiu s'ha fet un *snapshot [39]* d'un node *Slave* i s'ha guardat com a còpia de seguretat del sistema operatiu. El disc del sistema operatiu és idèntic a tots els nodes, així que amb un *snapshot* en tenim suficient. A més a més, com que l'*snapshot* és d'un node *Slave*, ens pot servir crear nous nodes i afegir-los en cas que ho necessitem.

6.1.4.2. Còpia de seguretat de les dades

Per mantenir les dades segures, s'han establert dos mètodes de còpia de seguretat.

El primer està basat en *barman.* Aquesta eina tampoc es troba als repositoris, així que l'hem afegit i l'hem instal·lat només al tercer node ja que només aquest node farà còpies de seguretat.

S'ha configurat el programa a partir del seu fitxer *barman.conf*, on s'ha especificat la informació necessària per accedir als tres nodes que formen el *clúster*. Un cop fet, s'ha disminuït la prioritat

del tercer node per a que es converteixi en node *Master*, de manera que les còpies afectin el mínim possible al rendiment.

Barman utilitza la funció de còpies de seguretat *PITR* de *PostgreSQL* nativament. De manera que cal fer una còpia de seguretat base, i a partir d'aquí mantenir tots els fitxers *WAL* per tal de poder reconstruir els canvis fins a un moment determinat del temps.

Per tal de fer això s'han hagut de modificar els fitxers *postgresql.conf* per a que enviïn els fitxers *WAL* a aquest servidor i així *barman* pugui anar emmagatzemant-los.

El segon mètode de còpia de seguretat està pensat només per a suplir les carències del primer mètode i s'utilitzarà puntualment. L'eina que es fa servir és *pg_dump* la qual fa un volcat d'una taula o base de dades a un fitxer de text.

Aquest mètode no permet la recuperació *PITR*, però sí que permet fer una còpia, esborrar la base de dades en qüestió i posteriorment restaurar-la a partir d'aquesta còpia. Cosa que és impossible amb la còpia PITR, ja que si es vol recuperar l'estat en un moment anterior del temps, es reverteixen tots els canvis de totes les bases de dades fins a aquell moment.

Per emmagatzemar la còpia de seguretat, s'ha afegit un nou disc amb el mateix format *LVM* de manera que el rendiment del sistema no es vegi gaire afectat.

Finalment quan aquestes còpies de seguretat han acabat, s'activa el programa que fa la còpia de seguretat a nivell de disc lògic del disc de còpies de seguretat. Intentant també, que l'impacte en el rendiment sigui baix.

6.1.5. Creació i automatització de procediments

Per a la correcta utilització de la solució i intentar reduir en major mesura l'error humà, s'han creat una vintena *d'scripts* que executen els procediments més bàsics. Tots els *scripts* apunten a un fitxer de configuració s*ettings.conf* el qual és llegit per alguns i modificat per altres. Els *scripts* s'han dividit en dos grups, els que s'executen automàticament i els que cal executar-los manualment, depenent de quin procediment es vulgui fer. El fitxer de configuració *settings.conf* junt amb tres *scripts* rellevants, com el d'habilitar els *daemons* de tot el *clúster*, el de convertir un node en *Master* i el de recuperació d'un node caigut els trobarem a l'annex.

6.1.5.1. Scripts automàtics

6.1.5.1.1. start_daemon.sh

Aquest *script* llegeix del fitxer de configuració si el *daemon* que resol els *failovers* s'hauria d'estar executant, i actua en conseqüència. Cal que el *daemon* sempre estigui iniciat, per això, aquest *script* s'executa cada minut (està al *cron* [40]).

6.1.5.1.2. auto_promote_node_failover.sh

L'executa el node *Slave* que s'ha de convertir en nou *Master* quan el node *Master* ha caigut. *L'script* és qui fa la conversió.

6.1.5.1.3. auto_follow_master_failovers.sh

L'executen tots els nodes (menys el nou node *Master*) quan hi ha hagut un *failover*. Executen una comanda la qual fa que el *PostgreSQL* es subscrigui a la *timeline* del nou node *Master*.

6.1.5.2. Scripts manuals

S'han creat també un conjunt de *scripts* els quals permeten el control tant de *PostgreSQL* com del *repmgr* de cada node del *clúster* per separat.

6.1.5.2.1. local_daemons_enable/disable/status.sh

Aquests tres *script* permeten habilitar, deshabilitar o consultat l'estat del *daemon* que resol els *failovers* de cada un dels nodes per separat. Quan s'habilita o es deshabilita el *daemon* es modifica el fitxer de configuració *settings.conf* per tal que *l'script start_daemon.sh* iniciï o no el *daemon*.

6.1.5.2.2. clúster_daemons_enable/disable/status.sh

S'ha vist la necessitat de poder executar els anteriors *scripts* de control del *daemon* de resolució de *failovers* a tots els nodes. Aquesta necessitat és la que solucionen aquests *scripts*. En algun procediment és necessari deshabilitar els *daemons* per evitar que s'executi el protocol de *failover* i posteriorment tornar-los a habilitar. Actualment només tenim tres nodes, però l'objectiu és que el sistema es pugui gestionar fàcilment encara que creixi.

6.1.5.2.3. start/stop/status/reload_postgres.sh

Les configuracions aplicades a *PostgreSQL* fan que aquest no es pugui controlar típicament des de */etc/init.d/postgresql*. Per facilitar l'administració s'han creat aquests tres *script* els quals llegeixen el fitxer *settings.conf* i utilitzen les eines de *PostgreSQL*.

6.1.5.2.4. first_clone_standby.sh

Quan afegim un nou node al *clúster*, primer de tot, caldrà copiar totes les dades a aquest nou node. Per fer-ho mantenint la coherència amb *PostgreSQL* cal primer fer la còpia de les dades amb una comanda específica de *PostgreSQL* i posteriorment aplicar tots els fitxers *WAL* que s'han generat des de l'inici de la còpia fins al moment de posta en marxa del node, amb una altra comanda *PostgreSQL*. Aquest procés es pot fer també amb la utilitat *repmgr* i això és el que implementa aquest *script*.

6.1.5.2.5. recover_with_rsync.sh

En cas de que caigui el node *Master* i un altre es converteixi en nou node *Master*, quan vulguem recuperar el node caigut no en tenim prou amb iniciar el *PostgreSQL* perquè les dades que aquest node té poden no estar actualitzades i per tant no podrem subscriure'l a la *timeline* del nou *Master*. Per aquest motiu s'ha creat aquest *script* el qual permet fer una còpia incremental de les dades actuals del *clúster*. D'aquesta manera el procés de recuperació d'un node és molt més ràpid que no pas fent la còpia completa.

6.1.5.2.6. register_as_master.sh

En cas de desastre ens pot interessar trencar el *clúster* i tornar-lo a muntar. En el moment que trenquem el *clúster*, caldrà tornar a registrar els nodes amb *repmgr* per tal de que aquest pugui resoldre els *failovers* correctament. Aquest *script* crea la base de dades i les taules necessàries perquè *repmgr* pugui funcionar correctament. També afegeix el node *Master* a la base de dades prèviament creada. Així que aquest *script* és possible que s'utilitzi sovint mentre el sistema estigui en desenvolupament, però un cop passi a producció només caldrà fer-lo servir en cas de desastre.

6.1.5.2.7. registre_as_slave.sh

Si es vol afegir un nou node al *clúster*, ja sigui perquè s'està recomponent el *clúster* després d'un desastre o si s'està engrandint, caldrà afegir el node al *repmgr* per tal de que aquest pugui reconèixer el node i tenir-lo en compte en cas de *failover.*

6.1.5.2.8. manual_promote_node.sh i manual_follow_master.sh

Aquesta parella *d'scripts* executen el mateix codi que executa el *repmgr* quan es dóna una situació de *failover*. El *repmgr* del node que es convertirà en *Master* executa el mateix codi que *manual_promote_node.sh* i els nodes que seguiran essent *Slaves*, executaran el mateix codi que *manual_follow_master.sh*.

Principalment els *scripts* es van crear per tal de simular situacions específiques i poder avaluar el comportament del *clúster*, però finalment s'han deixat operatius perquè s'ha vist que poden ser útils per a fer manteniment del *clúster*.

6.1.5.2.9. remove_node.sh

Per últim, és possible que el *clúster* necessiti engrandir-se, però també ho és que s'engrandeixi quan no toca i que posteriorment s'hagi de reduir. Per fer aquesta reducció caldrà executar aquest *script* des del node *Master,* el qual elimina el node seleccionat del *clúster*.

6.1.6. Creació de la documentació

Per tal de que l'equip pugui continuar desenvolupant i fent créixer aquest projecte s'ha anat documentant tot el projecte i a més a més s'ha creat procediments extra que caldrà executar en certes situacions. Els procediments documentats són sobre la *Second line*. Tot el que és referent a la *Front line* ja disposa de documentació de *F5* i un cop creada aquesta capa, no es necessitarà tocar més. Algun d'aquests procediments detallats els podrem trobar a l'annex II.

6.1.6.1. Afegir i eliminar un node al clúster

El fet d'afegir un node al *clúster* implica, en essència, afegir-lo a la *timeline* del node *Master,* registrar-lo al *repmgr*, afegir-lo a *barman* per tal de que si aquest node es converteix en *Master* pugui fer còpies de seguretat i finalment afegir-lo als *pools* de la *Front line*.

Per eliminar el node cal fer justament el contrari. Sobretot cal anar en compte de no activar el protocol de *failover* i per tant que es generin una sèrie de canvis dins el *clúster* els quals ens afecti en el procés d'eliminació.

6.1.6.2. Recuperar un node caigut

Com s'ha comentat anteriorment, recuperar un node caigut no és tant trivial com sembla ja que les dades que aquest node conté poden ser diferents a les del resta de nodes i per tant ha fet falta crear un procediment el qual es basa en executar alguns dels *scripts* anteriorment descrits.

6.1.6.3. Recuperació en cas de desastre (PITR)

Per recuperar tot el *clúster* en cas de desastre, tampoc és gens trivial ja que les còpies de seguretat estan a una màquina que no ens interessa que sigui node *Master,* així doncs, acord amb les necessitats de l'empresa a l'hora de donar el servei el bases de dades, s'ha redactat un procediment per a fer aquesta recuperació.

L'objectiu d'aquest procediment és que el sistema es pugui recuperar de la forma més fàcil possible. Per aquest motiu és inevitable convertir el tercer node en el node *Master* ja que és l'únic que té la còpia de seguretat i per tant l'únic que pot distribuir-les als altres nodes.

Això sí, aquest node només es manté com a *Master* mentre els altres nodes no estiguin completament sincronitzats, quan ho estiguin es forçarà una situació de *failover* per tal de que un altre node agafi el rol de *Master.*

6.2.Creació de la Front line

La creació de la *Front line* s'ha tractat com un petit projecte intern ja que s'havia de rebre suport de qui l'administra per tal de que l'actualització de versió no afectés als diferents serveis que depenen d'aquests balancejadors.

6.2.1. Actualització versió balancejadors F5

La versió mínima necessària pels balancejadors és la versió 10.5 però s'ha vist que actualment la versió més moderna és la 12.5, així que, per intentar que els balancejadors aguantin el màxim temps sense haver d'actualitzar-los altre cop, s'ha intentat posar la última versió.

La *Front line* està formada per un parell de balancejadors amb un *clúster* Actiu-Passiu, de manera que hi ha un dels balancejadors que tot i estar sincronitzat, no està donant servei. En aquest cas hi ha diferència entre els nodes i per tant parlarem de node principal i node secundari.

Primer de tot s'ha exportat tota la configuració per després poder-la tornar a carregar.

El protocol seguit per l'actualització ha estat obtenir les eines necessàries de la web oficial de *F5* i parar directament el node actiu (principal). D'aquesta manera el node inactiu (secundari) ha passat a estat actiu i hem pogut realitzar l'actualització correctament al node principal en la qual hem invertit 4 hores.

Un cop feta l'actualització s'ha trencat el *clúster* per tal de que el node secundari no sincronitzi les configuracions amb el node primari ja que tenen una versió diferent i el més probable es que la sincronització fallés i deixés coses a mig fer. S'ha importat tota la configuració i s'ha vist que hi havia un problema amb els certificats *SSL [41].* Bàsicament molts d'ells no es reconeixien correctament i per tant no es podia treballar amb ells.

Tot i buscar referències i posar-nos en contacte amb *F5* no hem pogut solucionar el problema, així que vam decidir repetir el procés d'actualització, però aquest cop posar la versió 11.5 que és la que ens van recomanar com a més estable.

Un cop repetit el procés d'actualització la importació de les configuracions va ser un èxit, aquest cop sense problemes de cap tipus.

Al cap d'un parell de dies vam repetir el procediment per actualitzar el node secundari, d'aquesta manera vam poder tornar a crear el *clúster*.

6.2.2. Creació dels pools

La creació dels pools és bastant senzilla, un cop es té el disseny de quins pools s'han de crear, el propi sistema dels balancejadors té un assistent que ajuda a crear-los. Això sí, per tal d'identificar quin node és el *Master* s'ha hagut de crear una vista a *PostgreSQL* la qual retorna *True* si efectivament aquell node és el *Master* o *False* si no ho és.

Aquesta vista mira directament la taula interna on PostgreSQL emmagatzema quins clients té subscrits a la seva *timeline*. Si no hi ha cap client subscrit, és perquè aquell node està subscrit a una altra *timeline* i per tant és un node *Slave*, per contra, si té subscriptors és que la *timeline* és seva i per tant és el *Master*.

L'altra vista que s'ha creat és per determinar si un node està actiu o no. Sempre retorna *True* a menys que el node no ho estigui que llavors retornarà un error. Els balancejadors són capaços de reconèixer l'error i per tant de saber que un node està actiu o inactiu.

Els balancejadors cada 10 segons consulten aquestes vistes i per tant tenen un control de qui és el *Master* i de quins nodes disposa el *clúster* en tot moment.

6.2.3. Creació de l'iRule

Arribem a l'últim punt de la implementació. Segons tota la documentació recollida a l'inici del projecte cal accedir a uns menús i a través d'un altre assistent establir el punt d'entrada de les noves regles que es volen afegir (la IP del balancejador i el port per defecte de *PostgreSQL*) i seleccionar quin tipus de base de dades es vol gestionar (*MySQL*, *PostgreSQL*, *MS SQL Server*, *Oracle*).

Això crea una entrada nova amb *l'iRule* associat al tipus de base de dades totalment preparat per a que modificant els pools i afegint el balanceig tot funcioni correctament.

Doncs en el moment de la implementació aquests menús amb les plantilles dels *iRules* pre-fets no els hem trobat. Després de buscar pels fòrums dels balancejadors vam trobar un *iRule* fet per un administrador del fòrum el qual gestionava bases de dades *MySQL*, però ni rastre de *l'iRule* per les bases de dades amb *PostgreSQL*.

Hem obert consulta amb els de *F5,* i het fer la resta d'implementacions mentre gestionàvem amb ells la incidència.

L'única conclusió que vam obtenir des de *F5* és que no tindríem disponible el fitxer *iRule* pre-fet dins la planificació marcada per acabar aquesta tasca, i possiblement tampoc abans d'acabar el projecte. Així que un cop acabades la resta d'implementacions, hem dedicat tant temps com hem pogut per crear *l'iRule* per nosaltres mateixos basant-nos amb el que havíem trobat per *MySQL*, però no hem estat capaços de completar aquesta part. Cal dir que desenvolupar aquest *iRule* mai havia estat dins la planificació ja que segons els manuals dels *F5* estava disponible des dels menús dels balancejadors.

7. Jocs de proves

Per validar la solució final s'han generat uns jocs de proves basats en *Pgbench [42]*, l'eina per mesurar el rendiment oferta per *PostgreSQL*. Segons la documentació de l'eina *[43]*, fàcilment es pot inicialitzar una base de dades i executar proves de rendiment sobre aquesta.

S'han dissenyat dos jocs de proves els quals ens permetran extreure unes conclusions sòlides sobre el guany o pèrdua de rendiment.

Creiem que és essencial mirar el rendiment tant quan les dades són a memòria com quan les dades són al disc, per aquest motiu, totes les proves treballaran amb dos conjunt de dades, un que hi càpiga a la memòria *RAM* i un altre que no hi càpiga.

Segons la configuració dels nodes, per comprovar la velocitat quan les dades són a la memòria ens cal inicialitzar una base de dades amb unes 7.000.000 files i per comprovar la velocitat quan les dades són a disc la quantitat de línies a crear a la base de dades poden ser de l'ordre de 60.000.000.

Cada joc de proves s'ha d'executar tant en el conjunt de la solució com en una instal·lació típica amb un sol servidor per tal de poder comparar els rendiments. A més a més, per obtenir unes dades fiables, els resultats mostrats s'extrauran a partir de la mitja de 5 execucions. Les dades amb les 5 execucions les trobarem a l'Annex III.

Aquest apartat mostrarà el disseny de les proves, el qual entenem que és independent a la seva execució, si bé, fins que no s'arregli la implementació de la *Front line*, no es podrà fer un estudi a fons del rendiment.

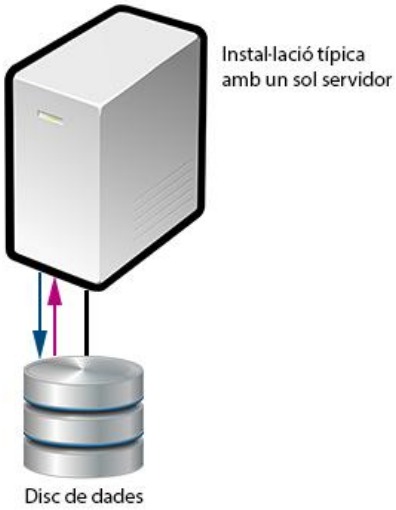


Figura 4: Representació d'una instal·lació típica amb un sol servidor

El mal funcionament de la capa balancejadora, ha impossibilitat fer aquests testos tal i com s'havien dissenyat inicialment, de manera que en comptes d'executar-los en el conjunt de la solució, s'han executat directament al node *Master* i s'ha comparat amb una instal·lació típica formada per un únic servidor.

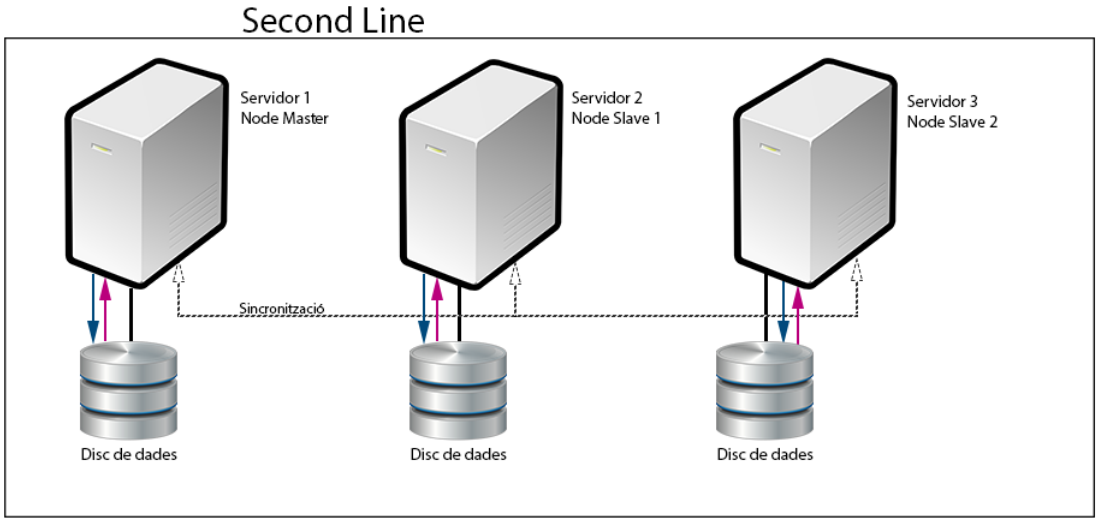


Figura 5: Representació de la Second Line

7.1. Test de lectures

7.1.1. Objectiu

Amb aquest joc de proves es vol analitzar l'escalat horitzontal de les lectures i quin *overhead [44]* afegeix la capa balancejadora.

Degut al problema amb la *Front line*, aquest objectiu s'ha vist substituït per: analitzar si hi ha guany en executar peticions de lectures en el node *Master* comparat amb una instal·lació típica.

7.1.2. Previsió de resultats

Sembla evident que el node *Master* resoldrà les peticions de lectura amb la mateixa velocitat que una instal·lació típica, ja que la nova funcionalitat aplicada a la *Second line (Streaming Replication)* només afecta a les escriptures i no a les lectures.

Per altra banda, com que l'accés a memòria és més ràpid que el de disc, creiem que les peticions que consultin dades que estan a memòria seran resoltes més ràpidament que no les que consulten dades de disc.

7.1.3. Proves

Per provar el rendiment de les lectures, executem aquests codis el qual inicialitzen la base de dades amb la mida adient i executa el test de rendiment.

```
#### TEST MEMORIA RAM ####
# Esborrem la base de dades (si existia)
dropdb bench1 --if-exists

# Creem la base de dades on farem les proves
createdb bench1

# Inicialitzem la base de dades amb 7.000.000 files
pgbench -i -s 70 bench1

# Executo test de només lectura amb 4 clients, 2 threads durant 10 minuts
pgbench -c 4 -j 2 -T 600 -S bench1
```

```
#### TEST DISC DUR ####
# Esborrem la base de dades (si existia)
dropdb bench1 --if-exists

# Creem la base de dades on farem les proves
createdb bench1

# Inicialitzem la base de dades amb 60.000.000 files
pgbench -i -s 600 bench1

# Executo test de només lectura amb 4 clients, 2 threads durant 10 minuts
pgbench -c 4 -j 2 -T 600 -S bench1
```

7.1.4. Resultats, interpretació i anàlisi

Els rendiments els hem mesurat amb *TPS's (Transactions Per Second) [45].* La mitja de les 5 execucions han estat les següents:

El node *Master* ha obtingut **3761,75** *TPS's* quan ha executat el primer test, el de memòria. Quan ha executat el test de disc, en canvi, ha obtingut **3605,46** *TPS's.*

El servidor amb la instal·lació típica ha obtingut valors molt semblants, ha obtingut **3752,96** *TPS's* quan ha executat el primer test, i **3636,30** *TPS's* quan ha executat el segon.

Com havíem previst, veiem que ambdós servidors tenen un rendiment semblant quan només executen lectures ja que les configuracions aplicades no afecten a les lectures. També podem veure que hi ha un increment de rendiment quan s'accedeix a dades que estan a memòria.

7.2. Test d'escriptures

7.2.1. Objectiu

En aquest test, es compararà el rendiment de la *Second line* amb una instal·lació típica quan ambdós executen peticions de només escriptura. L'objectiu d'aquest test és veure quina pèrdua de rendiment tenim quan executem aquest tipus de peticions.

Tot i que no es pugui executar el test en el conjunt del sistema, ens servirà per veure els *overheads* creats per la sincronització entre els nodes deguda al fet que han de mantenir la coherència.

7.2.2. Previsió de resultats

En aquest cas s'espera que el node *Master* tingui un pitjor rendiment que el servidor amb la instal·lació típica de *PostgreSQL.* Cada cop que s'executa una petició modificadora, el node *Master* ha d'enviar als altres nodes la modificació en qüestió per no perdre la coherència.

Tal i com s'implementa *l'Streaming Replication* el node *Master* ha d'enviar la modificació a tots els nodes, per tant, a mesura que s'incrementen els nodes, també s'incrementa el temps necessari per mantenir la coherència.

A diferència del test amb les lectures, el node *Master* ha de sincronitzar els canvis amb els altres nodes, per tant segurament veurem una caiguda en el rendiment.

7.2.3. Proves

Aquest cop volem provar el rendiment de les escriptures, per fer-ho creem una base de dades on farem les proves, la inicialitzem amb la mida adient i executem el test de rendiment.

```
#### TEST MEMORIA RAM ####
# Esborrem la base de dades (si existia)
dropdb bench1 --if-exists

# Creem la base de dades on farem les proves
createdb bench1

# Inicialitzem la base de dades amb 7.000.000 files
pgbench -i -s 70 bench1

# Executo test de només escriptura amb 4 clients, 2 threads durant 10 minuts
pgbench -c 4 -j 2 -T 600 -N bench1
```

```
#### TEST DISC DUR ####
# Esborrem la base de dades (si existia)
dropdb bench1 --if-exists

# Creem la base de dades on farem les proves
createdb bench1

# Inicialitzem la base de dades amb 60.000.000 files
pgbench -i -s 600 bench1

# Executo test de només escriptura amb 4 clients, 2 threads durant 10 minuts
pgbench -c 4 -j 2 -T 600 -N bench1
```

7.2.4. Resultats, interpretació i anàlisi

Un cop fetes les execucions hem fet les mitjanes amb els valors obtinguts, els resultats són els següents:

El node *Master* ha obtingut **668,97** *TPS's* quan ha executat el test de memòria. En canvi, quan ha executat el test de disc ha obtingut **518,14** *TPS's.*

El servidor amb la instal·lació típica aquest cop ha obtingut aconseguit un millor rendiment. Ha obtingut **720,93** *TPS's* quan ha executat el primer test, i **639,38** *TPS's* quan ha executat el segon.

Sembla que es compleix la previsió. El node *Master* ha de mantenir la coherència amb els altres nodes cada cop que fa una modificació, al fer-ho perd entre un 7% i un 19% de rendiment, depenent de si les dades estaven a memòria o a disc.

En un sistema en el qual la majoria de peticions siguin escripture aquest sistema no seria una bona opció ja que el temps necessari per resoldre cada modificació augmenta.

	Node *Master*		**Instal·lació típica**		**Guany**	
	Lectures	Escriptures	Lectures	Escriptures	Lectures	Escriptures
Memòria	3761,75 *TPS*	668.97 *TPS*	3752,96 *TPS*	720.93 *TPS*	-	**- 7.20 %**
Disc	3605,46 *TPS*	518,14 *TPS*	3636,3 *TPS*	639,38 *TPS*	-	**-18,97%**

Taula 3: Taula comparació dels resultats obtinguts ens les execucions

7.3. Test de connexions i contenció

Aquest test no s'ha pogut executar, ja que com descriu l'objectiu, cal executar-lo sobre la totalitat de la solució, i això no és possible ja que la *Front line* no és capaç de classificar les peticions ni balancejar-les.

7.3.1. Objectiu

L'últim joc de proves està enfocat a veure quin és el límit de càrrega que suporta la solució. Aquest joc de proves només s'executarà en la totalitat del sistema i no pas a un sol node ja que no té sentit fer un test d'estrès a un sol node perquè la solució està formada per dues capes i fent això només estaríem provant un node d'una capa en particular.

Amb aquest test i només ajustant els paràmetres, es pot extreure el llindar de càrrega que el sistema pot servir sense malmetre la qualitat del servei i per tant, quan caldria fer créixer el sistema afegint un nou node. Fins i tot es pot veure si és millor afegir un node o afegir recursos als nodes actuals. L'execució d'aquest test cal que sigui el més fidedigne possible, per tant, caldria executar-lo amb el major conjunt de dades, per tant, caldria inicialitzar la base de dades amb el segon conjunt de dades, el de 60.000.000 files.

7.3.2. Previsió dels resultats

Es faran 4 execucions que provaran diferents nivells de càrrega i contenció. S'espera que les tres primeres execucions no siguin un problema pel sistema, i que amb la quarta la càrrega de treball sigui superior al que el sistema pot donar. No es pot fer una estimació de quina càrrega és l'adequada per afegir un quart node ja que mai abans s'havien fet uns testos així dins l'empresa i per tant no hi ha punt de partida.

7.3.3. Proves

7.3.3.1. Prova pràcticament sense càrrega

```
#### TEST DISC DUR ####
# Esborrem la base de dades (si existia)
dropdb bench1 --if-exists

# Creem la base de dades on farem les proves
createdb bench1

# Inicialitzem la base de dades amb 60.000.000 files
pgbench -i -s 600 bench1

# Executo test amb només 1 client i 1 thread durant 10 minuts
# Situació "pràcticament sense càrrega"
pgbench -c 1 -T 600 bench1
```

7.3.3.2. Prova amb poca càrrega

```
#### TEST DISC DUR ####
# Esborrem la base de dades (si existia)
dropdb bench1 --if-exists

# Creem la base de dades on farem les proves
createdb bench1

# Inicialitzem la base de dades amb 60.000.000 files
pgbench -i -s 600 bench1


# Executo test amb 24 client 2 threads durant 10 minuts
# Situació de "poca càrrega i contenció"
pgbench -c 24 -j 2 -T 600 bench1
```

7.3.3.3. Prova amb càrrega i contenció elevada

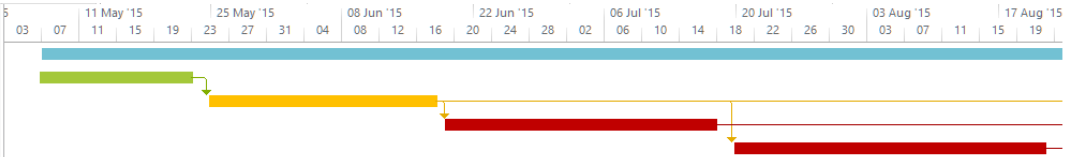
```
#### TEST DISC DUR ####
# Esborrem la base de dades (si existia)
dropdb bench1 --if-exists

# Creem la base de dades on farem les proves
createdb bench1

# Inicialitzem la base de dades amb 60.000.000 files
pgbench -i -s 600 bench1


# Executo test amb 64 clients i 4 threads durant 10 minuts
# Situació "càrrega i contenció elevada"
pgbench -c 64 -j 4 -T 600 bench1
```

7.3.3.4. Prova amb càrrega i contenció extrema

```
#### TEST DISC DUR ####
# Esborrem la base de dades (si existia)
dropdb bench1 --if-exists

# Creem la base de dades on farem les proves
createdb bench1

# Inicialitzem la base de dades amb 60.000.000 files
pgbench -i -s 600 bench1


# Executo test amb 192 clients i 6 threads durant 10 minuts
# Situació de "càrrega i contenció extrema"
pgbench -c 128 -j 6 -T 600 bench4
```

8. Planificació final

	Task Name	Duratic	Start	Finish
1	Projecte (Dates inicial I final)	148 days	Thu 07/05/15	Mon 30/11/15
2	Estudi de mercat i tria de la proposta	12 days	Thu 07/05/15	Fri 22/05/15
3	Construir els Cúster Master - Slave	18 days	Mon 25/05/15	Wed 17/06/15
4	Resolució automàtica de failovers	21 days	Fri 19/06/15	Fri 17/07/15
5	Còpies de seguretat	25 days	Mon 20/07/15	Fri 21/08/15
6	Capa balancejadora	31 days	Mon 12/10/15	Mon 23/11/15
7	Automatitzar procediments periòdics I facilitar l'administració (First line)	3 days	Tue 24/11/15	Thu 26/11/15
8	Automatitzar procediments periòdics I facilitar l'administració (Second line)	15 days	Mon 24/08/15	Fri 11/09/15
9	Documentació final (First line)	2 days	Fri 27/11/15	Mon 30/11/15
10	Documentació final (Second line)	8 days	Mon 14/09/15	Wed 23/09/15



	Task Name	Duratic	Start	Finish
1	Projecte (Dates inicial I final)	148 days	Thu 07/05/15	Mon 30/11/15
2	Estudi de mercat i tria de la proposta	12 days	Thu 07/05/15	Fri 22/05/15
3	Construir els Cúster Master - Slave	18 days	Mon 25/05/15	Wed 17/06/15
4	Resolució automàtica de failovers	21 days	Fri 19/06/15	Fri 17/07/15
5	Còpies de seguretat	25 days	Mon 20/07/15	Fri 21/08/15
6	Capa balancejadora	31 days	Mon 12/10/15	Mon 23/11/15
7	Automatitzar procediments periòdics I facilitar l'administració (First line)	3 days	Tue 24/11/15	Thu 26/11/15
8	Automatitzar procediments periòdics I facilitar l'administració (Second line)	15 days	Mon 24/08/15	Fri 11/09/15
9	Documentació final (First line)	2 days	Fri 27/11/15	Mon 30/11/15
10	Documentació final (Second line)	8 days	Mon 14/09/15	Wed 23/09/15

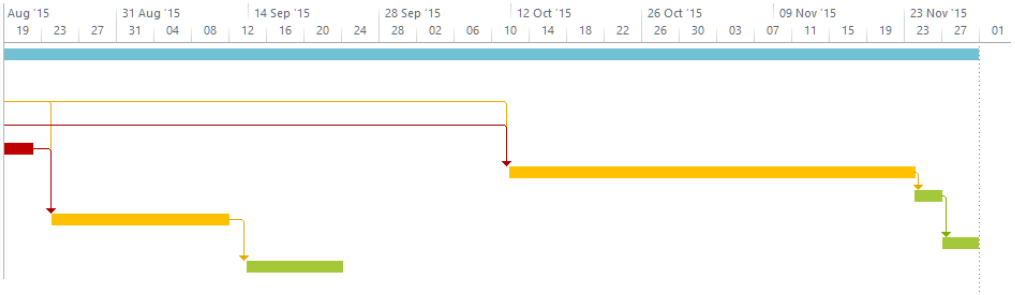


Figura 4: Gantt amb la planificació final del projecte

En colors podem veure la criticitat de les tasques: vermell > groc > verd

Degut a que la preparació dels nodes de la capa balancejadora, *Front line,* era un projecte intern, i que no se sabia exactament quan s'executaria, la planificació d'aquest projecte s'ha vist afectada ja que finalment s'ha hagut d'invertir més temps en la creació d'una *iRule* que no estava prevista haver de crear. Malgrat haver invertit el temps addicional per construir la nova *iRule*, aquesta no s'ha pogut completar. Tot i que no s'hagi aconseguit crear completament *l'iRule*, com podem veure a la *figura 4,* s'ha invertit més temps al projecte del que s'havia estimat, cosa que es veu reflectida en aquesta planificació.

9. Pressupost final

Els problemes trobats a la *Front line* per poder classificar les lectures i escriptures han fet que s'hagués d'invertir més temps al projecte del que s'havia planificat. Per la qual cosa s'ha vist modificat el pressupost. Només hi ha hagut una part del pressupost que es veiés afectada, però tot i això hi ha hagut una desviació la qual no es tenia planificada.

El cost final s'ha vist incrementat en **2920 €** per la desviació que s'ha tingut a l'hora d'implementar la capa balancejadora. Aquest cost ha vingut donat perquè finalment s'ha hagut de dedicar molt temps a implementar una part que no es tenia contemplat que s'hauria d'implementar, ja que a la documentació de *F5* semblava clar que aquesta part ja estava feta per ells.

9.1. Desviació capa balancejadora

Es va estimar que amb 6 hores hi hauria suficient temps per fer tot el procés d'actualització d'aquesta capa. Finalment, amb els problemes vistos per l'*SSL* a la versió 12.5, i haver d'instal·lar una altra versió inferior la qual no tingués aquests problemes, el temps dedicat a l'actualització ha estat de 17 hores.

Aquest increment ha estat degut, per una banda el fet d'haver actualitzat a la versió 12.5, trobar els problemes i intentar solucionar-los, i finalment posar la versió 11.5 la qual no va presentar aquests problemes.

Per altra banda, la documentació de *F5* no concorda amb els menús reals que hem trobat a l'hora d'identificar les peticions i balancejar-les i per tant les 24 hores que s'havia estimat en enllestir aquesta part, s'han convertit en 80 hores i aquesta part no s'ha pogut acabar.

La desviació d'aquest cost ha estat doncs d'un **323%**.

Tenint en compte aquestes noves dades el cost total del projecte ha estat de **18.181,81 €**. Que es veu desglossat de la següent manera.

9.2. Cost final

Concepte	**Unitats**	**Preu**	**Cost total**
Estudi de mercat	40 hores	40€ / hora	1.600,00 €
Muntatge del sistema			
Construcció del *clúster*			
Tècnic	45 hores	40€ / hora	1.800,00 €
Infraestructura	3 màquines durant 18 dies	25,53€ / mes	45,95 €
Resolució *failovers*			
Tècnic	55 hores	40€ / hora	2.200,00 €
Infraestructura	3 màquines durant 21 dies	25,53€ / mes	53,61 €
Còpies de seguretat			
Tècnic	100 hores	40€ / hora	4.000,00 €
Infraestructura	3 màquines durant 25 dies	38,16€ / mes	95,40 €
Capa balancejadora			
Tècnic	**97 hores**	**40€ / hora**	**3.880,00 €**
Automatització			
Tècnic	50 hores	40€ / hora	2.000,00 €
Infraestructura	3 màquines durant 18 dies	38,16€ / mes	68,69 €
Documentació			
Tècnic	40 hores	40€ / hora	1.600,00 €
Infraestructura	3 màquines durant 10 dies	38,16€ / mes	38,16 €
Migració de les bases de dades	20 hores	40€ / hora	800,00 €
TOTAL			**18.181,81 €**

La desviació final, doncs, ha estat del **17,28%**.

10. Conclusions

En aquest TFG hem vist el projecte de creació d'un sistema que ofereix un servei de bases de dades *PostgreSQL*. Aquest sistema està format per una primera capa on trobem un parell de balancejadors *F5* els quals identifiquen i classifiquen les peticions, i una segona capa on trobem tres nodes que les executen.

La segona capa ofereix un escalat horitzontal (en peticions de lectura) i ara per ara, l'execució de les peticions modificadores només la fa un dels nodes, el *Master*.

Malauradament no hem pogut fer les proves de rendiment que havíem dissenyat, de manera que no tenim un mecanisme per podem quantificar el guany obtingut. De fet, l'única cosa que hem pogut provar és que el temps necessari per resoldre una petició modificadora es veu incrementat respecte una instal·lació típica.

En el nostre cas rebem més peticions de lectura que d'escriptura, per tant, a l'hora de treballar amb un sistema com aquest, el qual balanceja les lectures, creiem que es podria obtenir un guany significatiu.

Per altra banda el que sí que hem pogut veure a les proves és que en un sistema on majoritàriament hi hagi escriptures, el rendiment del sistema podria ser pitjor que no pas el d'una instal·lació típica ja que al temps de resolució d'una petició modificadora, cal afegir el temps que necessita el node *Master* per distribuir els canvis i el temps que necessita el balancejador per identificar i classificar la petició.

En aquest cas caldria esperar-nos a que sortissin versions posteriors de *PostgreSQL* per muntar aquest sistema ja que segons la trajectòria de *PostgreSQL*, tot apunta a que les properes versions portaran una evolució de l'*Streaming Replication* la qual segurament ja podria tenir un sistema *Multi-Master.* D'aquesta manera el sistema també podria escalar, limitadament, les escriptures i per tant tenir un sistema molt més flexible.

El gran problema que hem trobat a l'hora de fer aquest projecte ha estat la diferència entre la teoria i la pràctica. És a dir, sorprenentment les eines fetes servir (*repmgr* i *barman*) les quals tenen una documentació més aviat pobre han estat les que no han presentat cap problema i han funcionat perfectament. En canvi on tenim més informació i documentació (configuració dels balancejadors) ha estat el que ha impossibilitat acabar el projecte dins el termini establert.

Això sí, que els balancejadors no els haguem pogut configurar correctament per falta de documentació o incoherència de la mateixa, no vol dir que aquest projecte hagi acabat, ni molt

menys. Realment, el que falta fer, comparat amb el que s'ha fet és poca cosa ja que només falta acabar la implementació de *l'iRule.* S'estima que amb unes 20 hores més es podria deixar el sistema plenament funcionant amb aquesta *iRule* implementada.

A més a més a l'empresa li interessa que es continuï indagant cap aquesta direcció ja que en un futur pròxim necessitaran un sistema que ofereixi, com a mínim, aquestes característiques.

11. Bibliografia

[1] Definició de RDS: https://en.wikipedia.org/wiki/Amazon_Relational_Database_Service

[2] Definició de DBMS: https://en.wikipedia.org/wiki/Database

[3] MySQL: https://www.mysql.com

[4] PostgreSQL: http://www.postgresql.org

[5] Definició de PITR: https://en.wikipedia.org/wiki/Point-in-time_recovery

[6] Pàgina oficial de Replication Manager: http://www.repmgr.org

Descripció extra de Replication Manager: http://2ndquadrant.com/es/recursos/repmgr

Codi a Github de Replication Manager: https://github.com/2ndQuadrant/repmgr

[7] Pàgina oficial de AWS RDS: https://aws.amazon.com/rds

[8] Definició de Cloud Computing: https://en.wikipedia.org/wiki/Cloud_computing

[9] Pàgina oficial d'AWS: https://aws.amazon.com

Descripció extra d'AWS: https://en.wikipedia.org/wiki/Amazon_Web_Services

[10] Pàgina oficial de Google Cloud: https://cloud.google.com

[11] Definició de WAL: http://www.postgresql.org/docs/9.4/static/wal-intro.html

Descripció extra de WAL: https://en.wikipedia.org/wiki/Write-ahead_logging

[12] Definició d'Streaming Replication: http://www.postgresql.org/docs/9.4/static/protocol-replication.html

Descripció extra d'Streaming Replication: https://wiki.postgresql.org/wiki/Streaming_Replication

[13] Pàgina oficial de Barman: http://www.pgbarman.org

Descripció extra de Barman: http://2ndquadrant.com/es/recursos/barman-gestor-de-respaldo-y-recuperacion-para-postgresql/

Codi a Github de Barman: https://github.com/2ndquadrant-it/barman

[14] Definició de Failover: https://en.wikipedia.org/wiki/Failover

[15] Llei Orgànica de Protecció de Dades https://es.wikipedia.org/wiki/Ley_Org%C3%A1nica_de_Protecci%C3%B3n_de_Datos_de_Car%C3%A1cter_Personal_de_Espa%C3%B1a

[16] Definició de metodologia Kanban: https://en.wikipedia.org/wiki/Kanban_(development)

[17] Pàgina oficial de Trello: https://trello.com/

[18] Definició de NAS: https://en.wikipedia.org/wiki/Network-attached_storage

[19] Programació conferencia 27/03/2015 PostgreSQL: http://www.pgconf.us/2015/schedule/

Esquema de la xerrada sobre Multi-Master: http://www.pgconf.us/2015/event/100/

[20] Pàgina oficial de 2nd Quadrant: http://2ndquadrant.com/en/

[21] Definició de DRBD: https://en.wikipedia.org/wiki/Distributed_Replicated_Block_Device

[22] Pàgina oficial de Slony: http://slony.info

Descripció extra de Slony: https://wiki.postgresql.org/wiki/Slony

[23] Pàgina oficial de pgpool-II: http://www.pgpool.net

Descripció extra de pgpool-II: https://wiki.postgresql.org/wiki/Pgpool-II

[24] Pàgina oficial de Bucardo: https://bucardo.org

Descripció extra de Bucardo: https://wiki.postgresql.org/wiki/Bucardo

[25] Pàgina comparació diferents solucions PostgreSQL: http://www.postgresql.org/docs/9.4/static/different-replication-solutions.html

[26] Definició de LTS: https://en.wikipedia.org/wiki/Long-term_support

[27] Definició d'ACL: https://en.wikipedia.org/wiki/Access_control_list

[28] Documentació pgdump: http://www.postgresql.org/docs/9.4/static/app-pgdump.html

[29] Descripció HTTP: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

[30] Descripció HTTPS: https://en.wikipedia.org/wiki/HTTPS

[31] Descripció pool: https://en.wikipedia.org/wiki/Pool_(computer_science)

[32] Documentació iRule: https://support.f5.com/kb/en-us/products/big-ip_ltm/manuals/product/bigip-system-irules-concepts-11-6-0.html

[33] Descripció proxy: https://en.wikipedia.org/wiki/Proxy_server

[34] Descripció RAID: https://en.wikipedia.org/wiki/RAID

[35] Descripció LVM: https://en.wikipedia.org/wiki/Logical_Volume_Manager_(Linux)

[36] Descripció timeline: http://www.postgresql.org/docs/9.4/static/continuous-archiving.html

[37] Definició rollback: http://www.postgresql.org/docs/9.4/static/sql-rollback.html

[38] Documentació continuous archiving: http://www.postgresql.org/docs/9.4/static/continuous-archiving.html

[39] Definició snapshot: https://en.wikipedia.org/wiki/Snapshot_(computer_storage)

[40] Descripció de cron: https://en.wikipedia.org/wiki/Cron

[41] Definició SSL: https://en.wikipedia.org/wiki/SSL

[42] Definició Pgbench: https://wiki.postgresql.org/wiki/Pgbench

[43] Documentació Pgbench: http://www.postgresql.org/docs/9.4/static/pgbench.html

[44] Definició d'overhead: https://en.wikipedia.org/wiki/Overhead_(computing)

[45] Definició TPS: https://en.wikipedia.org/wiki/Transactions_per_second

12. Annex I. Scripts i automatització

Aquest *script* conté tota la informació necessària per a que els altres *scripts* (tots criden aquest) funcionin correctament. En ell tant hi trobem les carpetes on s'emmagatzema les base de dades, es logs com el paràmetre que indica si el *daemon* del *repmgr* hauria d'estar engegat o no.

settings.conf

```
################## LOCAL VARIABLES  ##################

basic_path="/Dades"
scripts_path="$basic_path/manage"
this_server_name=`cat /etc/hostname | head -1`


############  POSTGRES REPMGR VARIABLES  ############

DB_path="$basic_path/data"
backup_path="$basic_path/backup"
logs_path="$basic_path/logs"

postgres_bin_path="/usr/lib/postgresql/9.4/bin"

repmgr_conf_path="$scripts_path/conf_scripts/repmgr.conf"
start_daemon_path="$scripts_path/conf_scripts/start_daemon.sh"

cluster_name=`grep cluster $repmgr_conf_path | cut -d '=' -f2`
repmgr_log="$logs_path/repmgrd.log"

auto_start_repmgrd=1

###################  SQL QUERIES  ###################

sql_query_node_names="SELECT name FROM repmgr_${cluster_name}.repl_nodes
WHERE cluster='$cluster_name'"
sql_query_node_type="SELECT type FROM repmgr_${cluster_name}.repl_nodes
WHERE cluster='$cluster_name' AND name='"$this_server_name"'"
sql_query_name_master="SELECT name FROM repmgr_${cluster_name}.repl_nodes
WHERE cluster='$cluster_name' AND type='master'"
sql_query_delete_node="DELETE FROM repmgr_${cluster_name}.repl_nodes WHERE
conninfo like " #node to remove


###################  FUNCTIONS  ###################

function info_print {
        d=`date "+%Y-%m-%d %H:%M:%S"`
        echo -e $d "-" $1
}

function error_print {
        d=`date "+%Y-%m-%d %H:%M:%S"`
        echo -e $d "- error: "$1
}

function send_mail {
        echo "send mail"
        #echo "$2" | mail -s "$1" root > /dev/null 2>&1
}
```

Aquí tenim la implementació que es fa servir per habilitar el *daemon* del *repmgr* a tots els servidors que formen el *clúster*. Agafa el nom dels servidors directament amb una consulta al *repmgr* i posteriorment s'hi connecta per habilitar l'inici automàtic (en cas que el *daemon* mori) i l'inicia perquè el canvi sigui efectiu immediatament i no quan s'executi l'script del *cron*.

cluster_daemons_enable.sh

```
#!/bin/bash

source "/Dades/manage/conf_scripts/settings.conf"

slaves=`repmgr -f $repmgr_conf_path cluster show | grep standby | awk
'{print $3}' | cut -d '=' -f2 | cut -d '-' -f1`

for server in $slaves; do
        info_print "enabling $server's autostart daemon"
        ssh -q $server -t "sed -i -e
's/auto_start_repmgrd=0/auto_start_repmgrd=1/g'
/Dades/manage/conf_scripts/settings.conf"
done

master=`repmgr -f $repmgr_conf_path cluster show | grep master | awk '{print
$4}' | cut -d '=' -f2 | cut -d '-' -f1`

for server in $master; do
        info_print "enabling $server's autostart daemon"
        ssh -q $server -t "sed -i -e
's/auto_start_repmgrd=0/auto_start_repmgrd=1/g'
/Dades/manage/conf_scripts/settings.conf"
done
```

Ens pot interessar poder convertir un node en *Master,* sobretot mentre el sistema es mantingui en desenvolupament. Això és el que fa aquest script, per executar-lo sense risc, cal haver deshabilitat els *daemons* del *repmgr* i haver parat el servidor *Master* ja que si no ho fem així tindrem dos *Masters* en el *clúster* i això pot tenir conseqüències que poden acabar amb pèrdua de dades.

manual_promote_node.sh

```
#!/bin/bash

source "/Dades/manage/conf_scripts/settings.conf"

info_print "Promoting server"
repmgr standby promote -f $repmgr_conf_path

if [ $? -ne 0 ]; then
        error_print "Promoting server"
        exit
fi

count=`ps aux | grep repmgrd | wc -l`

if [ $count -eq 2 ]; then
        info_print "Repmgr daemon is already running"
        exit 0
fi
```

```
info_print "Starting repmgr daemon"
repmgrd -f $repmgr_conf_path --verbose --monitoring-history --daemonize
if [ $? -ne 0 ]; then
        error_print "Starting repmgr daemon"
        exit 1
fi
```

La última implementació que veurem és la necessària per recuperar un node caigut (amb còpia incremental). Aquest script sincronitza les dades del node *Master* amb les dades del node en que l'executem. Per tenir el node completament funcional no n'hi ha prou amb executar aquest script, farà falta modificar manualment un fitxer de configuració i posteriorment tornar-lo a registrar a la base de dades del *repmgr*.

recover_with_rsync.sh

```
#!/bin/bash

source "/Dades/manage/conf_scripts/settings.conf"


if [ $# -ne 1 ]; then
        info_print "USAGE: $0 master-node"
        exit 1
fi

master=$1

status=`$postgres_bin_path/pg_ctl -D $DB_path status | grep "server is
running" | wc -l`


if [ $status -eq 1 ]; then

        info_print "Stopping PostgreSQL"
        $postgres_bin_path/pg_ctl -D $DB_path -m fast stop
else
        info_print "PostgreSQL already stopped"
fi

info_print "Updating from master"
repmgr -f $repmgr_conf_path --force --rsync-only -h $master -d repmgr -U
repmgr standby clone --verbose
if [ $? -ne 0 ]; then
        error_print "Updating from master"
        exit 1
fi
```

13. Annex II. Documentació de procediments

Recuperar node caigut de la *Second Line* (els tres nodes s'anomenen tuco1, tuco2 i tuco3).

Si un dels nodes del *cluster* cau, independentment de si era *Master* o *Slave* caldrà executar aquest procediment per tal de recuperar-lo com a nou ***Slave***. En aquest procediment, assumirem que el *Master* actual és **Tuco2**, i el node caigut que es vol recuperar és el node **Tuco1**.

1 - Primer de tot caldrà assegurar-nos que el *PostgreSQL* del node (tuco1) està parat (amb usuari postgres)

```
postgres@tuco1:~$
postgres@tuco1:~$ cd /Dades/manage/postgres/
### PostgreSQL iniciat
postgres@tuco2:/Dades/manage/postgres$ ./status_postgres.sh
2015-11-20 10:41:02 - Asking for PostgreSQL status
pg_ctl: server is running (PID: 52239)

### PostgreSQL parat
postgres@tuco1:/Dades/manage/postgres$ ./status_postgres.sh
2015-11-20 10:42:00 - Asking for PostgreSQL status
pg_ctl: server is not running
```

2 - Les dades que tindrà el node estaran des-actualitzades, per tant primer de tot caldrà actualitzar-les

```
postgres@tuco1:~$
postgres@tuco1:~$ cd /Dades/manage/repmgr/
### Especificar la IP de la interfície de xarxa eth1 del Master (tuco2)
postgres@tuco1:~$ ./recover_with_rsync.sh ip_node_master
```

3 - L'anterior punt copia tant les dades com les configuracions del node *Master,* per tant caldrà modificar el path d'on es guarden els *WAL*'s ja que cada node té un path diferent

Tots els servidors envien els archives al node *tuco3,* però es guarden en carpetes separades. Cal modificar doncs el camp *archive_command* del fitxer *postgresql.conf.* Després de la comanda anterior aquest paràmetre s'ha copiat de l'actual *Master*, per tant caldrà canviar-lo per la carpeta associada al node actual. Els paràmetres %p i %f els substitueix *PostgresSQL* automàticament.

archive_command ='chmod 666 %p && rsync -a %p tuco3-priv:/Dades/backup/tuco2/incoming/%f'
(copiat)

*archive_command = 'chmod 666 %p && rsync -a %p tuco3-priv:/Dades/backup/**tuco1**/incoming/%f'*
(modificat)

4 - Un cop fet això ja podem iniciar el procés *PostgreSQL* i verificar al log que s'ha iniciat com a *Slave*

```
postgres@tuco1:~$
postgres@tuco1:~$ cd /Dades/manage/postgres/
postgres@tuco1:/Dades/manage/postgres$ ./start_postgres.sh
2015-11-20 09:19:44 - Starting PostgreSQL
2015-11-20 09:19:44 - PostgreSQL started successfully

postgres@tuco1:~$ cd /Dades/logs
postgres@tuco1:/Dades/logs$ tail -10 postgresql-***.log
...
LOG:  started streaming WAL from primary at D6/63000000 on timeline 3
LOG:  consistent recovery state reached at D6/65000088
LOG:  database system is ready to accept read only connections
```

La *timeline* coincidirà amb la *timeline* dels altres nodes o rebem un error del tipus:
"FATAL: could not receive data from WAL

*stream: ERROR: requested WAL segment 00000003000000********** has already been removed"*

Si la *timeline* coincideix ja tenim el node recuperat, si ens apareix l'error, és que el node *Master* ha arxivat l'últim fitxer *WAL* el qual encara no s'havia transferit al node que estem recuperat.

Això ho podem fer recuperant el *WAL* a partir del gestor barman (veure el *man* de barman) o re-executant els passos 2- 4.

5 - Per últim només caldrà informar al *Replication Manager* de que el node que abans era *Master*, ara és un *Slave*, per fer-ho executarem el següent script.

```
postgres@tuco1:~$
postgres@tuco1:~$ cd /Dades/manage/repmgr/
postgres@tuco1:/Dades/manage/repmgr$ ./register_as_slave.sh

2015-11-20 09:20:31 - Registering as server
[2015-11-20 09:20:32] [NOTICE] standby node correctly registered for cluster pg_cluster with id 2
(conninfo: host=tuco2-priv user=repmgr dbname=repmgr)
```

Recuperació PITR en cas de desastre a la *Second Line* (els tres nodes s'anomenen tuco1, tuco2 i tuco3).

Aquest tipus de recuperació només caldrà fer-la en cas de desastre, ja sigui corrupció de dades o crash del *clúster*. La recuperació **esborrarà** les bases de dades i configuracions de PostgreSQL a més aquesta recuperació cal fer-la des del servidor *Master* i amb especial compte en l'ordre dels punts del procediment.

En el punt 5 caldrà parar el servei del servidor *Master* i és molt important que els *daemons* que resolen els *failovers* estiguin deshabilitats a la resta del *clúster* ja que si no ho estan saltarà el procediment de *failover* i es promocionarà un altre servidor com a nou *Master* i per tant caldrà re-començar el procés considerant el nou *Master.*

En el exemples se suposarà que el *Master* és tuco1 i que tuco3 és el servidor gestor dels backups i és *Slave*:

1 - Primer de tot caldrà que parem els *daemons* per poder treballar sense riscs de que en cas d'error mentre executem el procediment, s'executi el protocol de failover. Per fer això, podem dirigir-nos a */Dades/manage/daemon/* del *Master* i executar l'script *cluster_daemons_disable.sh* amb usuari *postgres*:

```
postgres@tuco1:/Dades/manage/daemon$ ./cluster_daemons_disable.sh
2015-10-01 12:15:45 - disabling tuco2's autostart daemon
2015-10-01 12:15:45 - disabling tuco3's autostart daemon
2015-10-01 12:15:45 - disabling tuco1's autostart daemon
2015-10-01 12:15:46 - stopping tuco2's daemon
2015-10-01 12:15:47 - stopping tuco3's daemon
2015-10-01 12:15:47 - stopping tuco1's daemon
```

2 - Seguidament caldrà que anem al servidor que s'encarrega de gestionar els backups (tuco3) i parem el PostgreSQL, esborrem tot el contingut actual de la base de dades i recuperem el backup que ens interessa.

2.1 - Procedim doncs a parar el servei de PostgreSQL del servidor tuco3. Per fer-ho ens dirigirem a la carpeta */Dades/manage/postgres*, on hi ha els *scripts* per controlar aquest servei i executarem *l'script stop_postgres.sh.*

```
postgres@tuco3:/Dades/manage/postgres$ ./stop_postgres.sh
2015-10-08 11:54:17 - Stopping PostgreSQL
waiting for server to shut down.... done
server stopped
2015-10-08 11:54:18 - PostgreSQL stopped succesfull
2015-10-01 12:20:17 - Stopping PostgreSQL
waiting for server to shut down.... done
server stopped
2015-10-01 12:20:18 - PostgreSQL stopped succesfull
```

2.2 - Per comprovar de quins backups disposem, podem fer servir la següent comanda amb usuari **barman**:

```
barman@tuco3:~$ barman list-backup tuco1
tuco1 20151001T020003 - Thu Oct  1 02:00:12 2015 - Size: 32.6 MiB - WAL Size: 14.2 MiB
tuco1 20150930T140002 - Wed Sep 30 14:00:09 2015 - Size: 32.6 MiB - WAL Size: 18.4 MiB
```

En aquest cas, veiem que disposem de 2 backups, un del 30/09/2015 a les 14:00:09 i un altre el 01/09/2015 a les 2:00:12.

Els backups consten de dues parts:

- Un *base backup* el qual és un còpia de tota la carpeta de bases de dades i configuracions
- Els *archives,* que *serveixen* per poder recuperar l'estat de la base de dades des del moment en que s'ha fet el backup fins el següent backup o si no hi ha següent backup, fins al moment actual.

El backup amb identificador *20150930T140002* té el base backup del 30/09/2015 a les 14:00:09 i *archives* fins el següent backup (*20151001T020003*), per tant aquest backup permet recuperar l'estat de la base de dades des del 30/09/2015 a les 14:00:09 fins el 01/09/2015 a les 02:00:11

El backup amb identificador *20151001T020003* té el *base backup* del 01/09/2015 a les 02:00:12 i com que no hi ha backups més recents, permet recuperar fins el moment actual.

2.3 - Podem obtenir més informació sobre el backup en qüestió executant la següent comanda:

```
barman@tuco3:~$ barman show-backup tuco1 20151001T020003
Backup 20151001T020003:
Server Name          : tuco1
Status               : DONE
PostgreSQL Version   : 90404
PGDATA directory     : /Dades/data

Base backup information:
Disk usage           : 32.6 MiB (32.6 MiB with WALs)
Incremental size     : 274.6 KiB (-99.18%)
Timeline             : 7
Begin WAL            : 000000070000004C00000073
End WAL              : 000000070000004C00000073
WAL number           : 1
WAL compression ratio: 99.84%
Begin time           : 2015-10-01 02:00:03.912651+02:00
End time             : 2015-10-01 02:00:12.968206+02:00
Begin Offset         : 200
End Offset           : 456
Begin XLOG           : 4C/730000C8
End XLOG             : 4C/730001C8

WAL information:
No of files          : 574
Disk usage           : 14.8 MiB
WAL rate             : 1616138.60/hour
Compression ratio    : 99.84%
Last available       : 000000070000004E000000B1

Catalog information:
Retention Policy     : not enforced
Previous Backup      : 20150930T140002
Next Backup          : - (this is the latest base backup)
```

2.4 - Per tal de recuperar-lo, caldrà eliminar el contingut de la carpeta */Dades/data* i canviar el propietari a usuari barman, perquè aquest pugui accedir -hi.

```
root@tuco3:~# rm /Dades/data/* -Rf
root@tuco3:~# chown barman:barman /Dades/data –R
```

(En el cas de que es vulgui recuperar en una altre carpeta diferent, cal crear-la en un lloc que hi hagi suficient espai per emmagatzemar la còpia i que barman tingui accés. I seguir el procediment canviant la carpeta destí)

2.5 - A continuació caldrà indicar-li a barman que volem que recuperi el backup especificat:

```
barman@tuco3:~$ barman recover tuco1 20151001T020003 /Dades/data
Starting local restore for server tuco1 using backup 20151001T020003
Destination directory: /Dades/data
Copying the base backup.
Copying required wal segments.
Generating archive status files
Disabling dangerous settings in destination directory.
The archive_command was set to 'false' to prevent data losses.

Your PostgreSQL server has been successfully prepared for recovery!

Please review network and archive related settings in the PostgreSQL
configuration file before starting the just recovered instance.

Recovery completed successful.
```

*2.6 - Un cop acabada la recuperació tornarem a posar l'usuari **postgres** com a propietari de la carpeta /Dades/data.*

```
root@tuco3:~# chown postgres:postgres /Dades/data/ -R
```

3 - Acte seguit, modificarem el postgresql.conf del backup que acabem de recuperar i especificarem la hora a la qual es vol tornar.

3.1 - Les línies que cal modificar al fitxer postgresql.conf s'utilitzen per l'*archive*, cada servidor guarda els seus *archives* a carpetes diferents i com que el backup l'hem extret d'un servidor diferent al que el carregarem, cal canviar-ho:

```
...
#archive_command =  'chmod 666 %p && cp -i %p /Dades/backup/tuco3/incoming/%f'
```

```
#BARMAN# archive_command = 'chmod 666 %p && rsync -a %p tuco3-
priv:/Dades/backup/tuco1/incoming/%f'
archive_command = false
...
```

Aquestes línies cal canviar-les per que quedin de la següent manera:

```
...
archive_command =  'chmod 666 %p && cp -i %p /Dades/backup/tuco3/incoming/%f'
#archive_command = 'chmod 666 %p && rsync -a %p tuco3-
priv:/Dades/backup/tuco1/incoming/%f'
...
```

Hem descomentat la primera línia, hem preparat la del mig traient el comentari inicial i hem eliminat la última.

3.2 - Per especificar l'hora caldrà crear el fitxer *recovery.conf* dins la carpeta on hem recuperat el backup on especificarem l'hora a la que volem tornar, per exemple el 01/10/2015 a les 11:30:02:

```
barman@tuco3:~$ echo "recovery_target_time = '2015-10-01 11:30:02'"
> /Dades/data/recovery.conf
```

Si volem recuperar just al moment després posarem recovery_target_inclusive = true.
En canvi si volem just el moment anterior, posarem recovery_target_inclusive = false.

```
barman@tuco3:~$ echo "recovery_target_inclusive = true" >> /Dades/data/recovery.conf
```

Per últim indiquem que no hi ha comanda per recuperar els archives ja que barman ja els ha posat a la carpeta pg_xlog.

```
barman@tuco3:~$ echo "restore_command = '"' >> /Dades/data/recovery.conf
```

4 - Iniciem el servei, li indiquem a postgres que volem recuperar el backup i revisem que al log (/Dades/logs/postgres*.log) confirmi que la recuperació s'ha fet correctament.**

4.1 - El contingut del fitxer de log hauria de ser de l'estil del següent:

```
LOG:  starting point-in-time recovery to 2015-10-01 11:30:02+02
LOG:  redo starts at 4C/730000C8
LOG:  consistent recovery state reached at 4C/730001C8
...
```

```
LOG:  database system is ready to accept read only connections
LOG:  recovery stopping before commit of transaction 36131, time 2015-10-01
12:02:37.980032+02
LOG:  recovery has paused
HINT:  Execute pg_xlog_replay_resume() to continue.
```

4.2 - Caldrà aplicar els canvis que hi ha als *WALS*, amb usuari **postgres,** de la següent manera i la sortida hauria de ser de l'estil següent:

```
postgres@tuco3:/~# echo "SELECT pg_xlog_replay_resume();" | psql
pg_xlog_replay_resume
----------------------
(1 row)
```

Veurem que al log han aparegut les següents línies, les quals ens indicaran que la recuperació ha acabat correctament:

```
LOG:  redo done at 4E/CB000388
LOG:  selected new timeline ID: 8
LOG:  archive recovery complete
LOG:  MultiXact member wraparound protections are now enabled
LOG:  database system is ready to accept connections
LOG:  autovacuum launcher started
```

Com que el servidor **tuco1** era el *Master*, ara tuco3 també és *Master*, però ho és d'una *timeline* diferent, no interfereix en l'*Streaming Replication* entre **tuco1 i tuco2**.
A partir d'aquest moment, tuco3 ja pot servir peticions tant de lectura com d'escriptura.
5 - Ara tenim 2 *Masters* però poden conviure sense interferir en el servei, caldrà doncs migrar tuco1 i tuco2 a la nova *timeline* que manté tuco3 i assegurar la coherència entre el repmgr i PostgreSQL.

5.1 - Caldrà parar tuco1 i tuco2, accedir a /Dades/manage/repmgr i executar *l'script* first_clone_standby.sh **per cada un d'ells**. Aquest *script* esborrarà les dades de PostgreSQL i carregarà el backup recuperat des de tuco3.

```
postgres@tuco3:/Dades/manage/repmgr#./first_clone_standby.sh
...
[2015-10-01 12:40:50] [INFO] executing: '/usr/lib/postgresql/9.4/bin/pg_basebackup -l "repmgr base
backup"  -h tuco3-priv -p 5432 -U repmgr -D /Dades/data
NOTICE:  pg_stop_backup complete, all required WAL segments have been archived
```

```
[2015-10-01 12:42:54] [NOTICE] standby clone (using pg_basebackup) complete
[2015-10-01 12:42:54] [NOTICE] HINT: you can now start your PostgreSQL server
[2015-10-01 12:42:54] [NOTICE] for example : pg_ctl -D /Dades/data start
```

5.2 - Modificarem el fitxer */Dades/data/postgresql.conf* per indicar on volem que aquests servidors guardin els fitxers *WAL*, cal canviar aquestes línies:

```
...
archive_command =  'chmod 666 %p && cp -i %p /Dades/backup/tuco3/incoming/%f'
#archive_command = 'chmod 666 %p && rsync -a %p tuco3-priv:/Dades/backup/tuco1/incoming/%f'
...
```

per aquestes:

Aquestes línies cal canviar-les per que cada un dels servidors apunti a la seva carpeta.

```
tuco1 -> tuco3-priv:/Dades/backup/tuco1/incoming/%f
tuco2 -> tuco3-priv:/Dades/backup/tuco2/incoming/%f
...
#archive_command =  'chmod 666 %p && cp -i %p /Dades/backup/tuco3/incoming/%f'
…
archive_command = 'chmod 666 %p && rsync -a %p tuco3-priv:/Dades/backup/tuco1/incoming/%f'
...
```

5.3 - *Iniciem els serveis de les dues màquines de la següent manera*

```
postgres@tuco1:/Dades/manage/postgres$ ./start_postgres.sh
2015-10-01 12:44:15 - Starting PostgreSQL
server starting
2015-10-01 12:44:19 - PostgreSQL started successfully
postgres@tuco1:/Dades/manage/postgres$ LOG:  redirecting log output to logging collector process
HINT:  Future log output will appear in directory "/Dades/logs".
```

Si verifiquem al fitxer /Dades/logs, veurem que el servei s'ha iniciat correctament i que ja són a la mateixa timeline 1, la mateixa que tuco3, i per tant, l'Streaming Replication ja funciona correctament:

```
...
LOG:  entering standby mode
LOG:  started streaming WAL from primary at 2/8F000000 on timeline 1
LOG:  redo starts at 2/8F0000C8
LOG:  consistent recovery state reached at 2/920000F0
…
```

```
LOG:  database system is ready to accept read only connections
```

5.4 - Un cop arribem aquí tindrem la replicació perfectament funcionant, però la base de dades del repmgr no és coherent amb la nova realitat, ja que ***hem restaurat la base de dades a quan tuco1 era Master, però ara el Master és tuco3****.*

Per solucionar aquesta situació, el més fàcil és que entrem amb pgadmin a tuco3, esborrem l'schema ***repmgr_pg_cluster*** *(en cascada) i el tornem a crear (com s'explica a continuació).*

Un cop eliminat, des de tuco3 executem l'script ***/Dades/manage/repmgr/register_as_master.sh*** *el qual crearà altre cop l'schema on tuco3 serà el master.*

```
postgres@tuco3:/Dades/manage/repmgr$ ./register_as_master.sh
[2015-10-01 12:50:36] [NOTICE] master node correctly registered for cluster pg_cluster with id 1
(conninfo: host=tuco1-priv user=repmgr dbname=repmgr)
```

Als a tots els altres nodes, caldrà que executem l'script ***/Dades/manage/repmgr/register_as_slave.sh*** *per tal d'afegir els nodes al clúster.*

```
postgres@tuco2:/Dades/manage/repmgr$ ./register_as_slave.sh
2015-10-01 12:50:50 - Registering as slave
[2015-10-01 12:50:50] [NOTICE] standby node correctly registered for cluster pg_cluster with id 2
(conninfo: host=tuco2-priv user=repmgr dbname=repmgr)
...
```

5.5 - Ara ja ho tenim tot correctament, funcionant, així que ens podem dirigir ja al nou Master (tuco3) i tornar a habilitar els daemon del replication manager amb la següent comanda:

```
postgres@tuco3:/Dades/manage/daemon$ ./cluster_daemons_enable.sh
2015-10-01 12:56:50 - enabling tuco2's autostart daemon
2015-10-01 12:56:50 - enabling tuco1's autostart daemon
2015-10-01 12:56:51 - enabling tuco3's autostart daemon
```

6.0 - (Opcional, però recomanable) Amb aquest procediment, tindrem com a nou Master el servidor de gestió de backup, així que és recomanable que promocionem a un altre node com a Master i aquest el convertim en Slave.

6.1 - Això ho podem fer simplement, parant el servei a tuco3:

```
postgres@tuco3:/Dades/manage/postgres$ ./stop_postgres.sh
2015-10-08 11:54:17 - Stopping PostgreSQL
waiting for server to shut down.... done
server stopped
2015-10-08 11:54:18 - PostgreSQL stopped succesfull
2015-10-01 13:00:17 - Stopping PostgreSQL
waiting for server to shut down.... done
server stopped
2015-10-01 13:00:18 - PostgreSQL stopped succesfull
```

6.2 - Esperarem que es resolgui el failover. (Podem comprovar-ho amb l'script /Dades/manage/show_cluster.sh).

```
postgres@tuco1:/Dades/manage$ ./show_cluster.sh
Role      | Connection String
...
FAILURE   | host=tuco3-priv user=repmgr dbname=repmgr
standby | host=tuco2-priv user=repmgr dbname=repmgr
…
* master  | host=tuco1-priv user=repmgr dbname=repmgr
```

6.3 - Finalment recuperem el servidor tuco3 com a Slave:

```
postgres@tuco3:/Dades/manage/repmgr$ ./recover_with_rsync.sh
...
[2015-10-01 12:42:54] [NOTICE] standby clone (using pg_basebackup) complete
[2015-10-01 12:42:54] [NOTICE] HINT: you can now start your PostgreSQL server
[2015-10-01 12:42:54] [NOTICE] for example : pg_ctl -D /Dades/data start
```

14. Annex III. Execucions dels jocs de proves

Execucions amb peticions de només lectura al node *Master,* accedint a la memòria

```
postgres@tuco1:~$ dropdb bench1 --if-exists
postgres@tuco1:~$ createdb bench1
postgres@tuco1:~$ pgbench -i -s 70 bench1

postgres@tuco1:~$ pgbench -c 4 -j 2 -T 600 -S bench1
starting vacuum...end.
transaction type: SELECT only
scaling factor: 70
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 1985462
latency average: 1.209 ms
tps = 3309.032812 (including connections establishing)
tps = 3309.236274 (excluding connections establishing)
```

```
postgres@tuco1:~$ dropdb bench1 --if-exists
postgres@tuco1:~$ createdb bench1
postgres@tuco1:~$ pgbench -i -s 70 bench1

postgres@tuco1:~$ pgbench -c 4 -j 2 -T 600 -S bench1
starting vacuum...end.
transaction type: SELECT only
scaling factor: 70
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 1990702
latency average: 1.206 ms
tps = 3317.816102 (including connections establishing)
tps = 3317.996851 (excluding connections establishing)
```

```
postgres@tuco1:~$ dropdb bench1 --if-exists
postgres@tuco1:~$ createdb bench1
postgres@tuco1:~$ pgbench -i -s 70 bench1

postgres@tuco1:~$ pgbench -c 4 -j 2 -T 600 -S bench1
starting vacuum...end.
transaction type: SELECT only
scaling factor: 70
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 2253234
latency average: 1.065 ms
tps = 3755.330991 (including connections establishing)
tps = 3755.470683 (excluding connections establishing)
```

```
postgres@tuco1:~$ dropdb bench1 --if-exists
postgres@tuco1:~$ createdb bench1
postgres@tuco1:~$ pgbench -i -s 70 bench1

postgres@tuco1:~$ pgbench -c 4 -j 2 -T 600 -S bench1
starting vacuum...end.
transaction type: SELECT only
scaling factor: 70
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 2505062
latency average: 0.958 ms
tps = 4175.089660 (including connections establishing)
tps = 4175.238507 (excluding connections establishing)
```

```
postgres@tuco1:~$ dropdb bench1 --if-exists
postgres@tuco1:~$ createdb bench1
postgres@tuco1:~$ pgbench -i -s 70 bench1

postgres@tuco1:~$ pgbench -c 4 -j 2 -T 600 -S bench1
starting vacuum...end.
transaction type: SELECT only
scaling factor: 70
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 2550918
latency average: 0.941 ms
tps = 4251.490716 (including connections establishing)
tps = 4251.745323 (excluding connections establishing)
```

Execucions amb peticions de només lectura del servidor amb la instal·lació típica, accedint a la memòria

```
postgres@test0:~$ dropdb bench1 --if-exists
postgres@test0:~$ createdb bench1
postgres@test0:~$ pgbench -i -s 70 bench1

postgres@test0:~$ pgbench -c 4 -j 2 -T 600 -S bench1
starting vacuum...end.
transaction type: SELECT only
scaling factor: 70
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 2012781
latency average: 1.163 ms
tps = 3354.451248 (including connections establishing)
tps = 3354.845175 (excluding connections establishing)
```

```
postgres@test0:~$ dropdb bench1 --if-exists
postgres@test0:~$ createdb bench1
postgres@test0:~$ pgbench -i -s 70 bench1

postgres@test0:~$ pgbench -c 4 -j 2 -T 600 -S bench1
starting vacuum...end.
transaction type: SELECT only
scaling factor: 70
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 2001284
latency average: 1.197 ms
tps = 3335.221547 (including connections establishing)
tps = 3335.735641 (excluding connections establishing)
```

```
postgres@test0:~$ dropdb bench1 --if-exists
postgres@test0:~$ createdb bench1
postgres@test0:~$ pgbench -i -s 70 bench1

postgres@test0:~$ pgbench -c 4 -j 2 -T 600 -S bench1
starting vacuum...end.
transaction type: SELECT only
scaling factor: 70
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 2154819
latency average: 1.113 ms
tps = 3591.112584 (including connections establishing)
tps = 3591.494175 (excluding connections establishing)
```

```
postgres@test0:~$ dropdb bench1 --if-exists
postgres@test0:~$ createdb bench1
postgres@test0:~$ pgbench -i -s 70 bench1

postgres@test0:~$ pgbench -c 4 -j 2 -T 600 -S bench1
starting vacuum...end.
transaction type: SELECT only
scaling factor: 70
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 2535260
latency average: 0.951 ms
tps = 4225.204163 (including connections establishing)
tps = 4225.524072 (excluding connections establishing)
```

```
postgres@test0:~$ dropdb bench1 --if-exists
postgres@test0:~$ createdb bench1
postgres@test0:~$ pgbench -i -s 70 bench1

postgres@test0:~$ pgbench -c 4 -j 2 -T 600 -S bench1
starting vacuum...end.
transaction type: SELECT only
scaling factor: 70
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 2555424
latency average: 0.936 ms
tps = 4258.816016 (including connections establishing)
tps = 4259.235842 (excluding connections establishing)
```

Execucions amb peticions de només lectura al node *Master,* accedint al disc

```
postgres@tuco1:~$ dropdb bench1 --if-exists
postgres@tuco1:~$ createdb bench1
postgres@tuco1:~$ pgbench -i -s 600 bench1

postgres@tuco1:~$ pgbench -c 4 -j 2 -T 600 -S bench1
starting vacuum...end.
transaction type: SELECT only
scaling factor: 600
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 1989568
latency average: 1.216 ms
tps = 3315.795186 (including connections establishing)
tps = 3316.158312 (excluding connections establishing)
```

```
postgres@tuco1:~$ dropdb bench1 --if-exists
postgres@tuco1:~$ createdb bench1
postgres@tuco1:~$ pgbench -i -s 600 bench1

postgres@tuco1:~$ pgbench -c 4 -j 2 -T 600 -S bench1
starting vacuum...end.
transaction type: SELECT only
scaling factor: 600
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 2157812
latency average: 1.088 ms
tps = 3596.218465 (including connections establishing)
tps = 3596.854796 (excluding connections establishing)
```

```
postgres@tuco1:~$ dropdb bench1 --if-exists
postgres@tuco1:~$ createdb bench1
postgres@tuco1:~$ pgbench -i -s 600 bench1

postgres@tuco1:~$ pgbench -c 4 -j 2 -T 600 -S bench1
starting vacuum...end.
transaction type: SELECT only
scaling factor: 600
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 2179875
latency average: 1.083 ms
tps = 3632.894587 (including connections establishing)
tps = 3633.300412 (excluding connections establishing)
```

```
postgres@tuco1:~$ dropdb bench1 --if-exists
postgres@tuco1:~$ createdb bench1
postgres@tuco1:~$ pgbench -i -s 600 bench1

postgres@tuco1:~$ pgbench -c 4 -j 2 -T 600 -S bench1
starting vacuum...end.
transaction type: SELECT only
scaling factor: 600
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 2194216
latency average: 1.069 ms
tps = 3656.887453 (including connections establishing)
tps = 3657.167842 (excluding connections establishing)
```

```
postgres@tuco1:~$ dropdb bench1 --if-exists
postgres@tuco1:~$ createdb bench1
postgres@tuco1:~$ pgbench -i -s 600 bench1

postgres@tuco1:~$ pgbench -c 4 -j 2 -T 600 -S bench1
starting vacuum...end.
transaction type: SELECT only
scaling factor: 600
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 2295452
latency average: 0.989 ms
tps = 3825.517246 (including connections establishing)
tps = 3825.995844 (excluding connections establishing)
```

Execucions amb peticions de només lectura del servidor amb la instal·lació típica, accedint al disc

```
postgres@test0:~$ dropdb bench1 --if-exists
postgres@test0:~$ createdb bench1
postgres@test0:~$ pgbench -i -s 600 bench1

postgres@test0:~$ pgbench -c 4 -j 2 -T 600 -S bench1
starting vacuum...end.
transaction type: SELECT only
scaling factor: 600
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 2010157
latency average: 1.112 ms
tps = 3350.001945 (including connections establishing)
tps = 3350.312281 (excluding connections establishing)
```

```
postgres@test0:~$ dropdb bench1 --if-exists
postgres@test0:~$ createdb bench1
postgres@test0:~$ pgbench -i -s 600 bench1

postgres@test0:~$ pgbench -c 4 -j 2 -T 600 -S bench1
starting vacuum...end.
transaction type: SELECT only
scaling factor: 600
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 2084242
latency average: 1.098 ms
tps = 3473.518526 (including connections establishing)
tps = 3474.020437 (excluding connections establishing)
```

```
postgres@test0:~$ dropdb bench1 --if-exists
postgres@test0:~$ createdb bench1
postgres@test0:~$ pgbench -i -s 600 bench1

postgres@test0:~$ pgbench -c 4 -j 2 -T 600 -S bench1
starting vacuum...end.
transaction type: SELECT only
scaling factor: 600
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 2185424
latency average: 1.081 ms
tps = 3642.231851 (including connections establishing)
tps = 3642.549587 (excluding connections establishing)
```

```
postgres@test0:~$ dropdb bench1 --if-exists
postgres@test0:~$ createdb bench1
postgres@test0:~$ pgbench -i -s 600 bench1

postgres@test0:~$ pgbench -c 4 -j 2 -T 600 -S bench1
starting vacuum...end.
transaction type: SELECT only
scaling factor: 600
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 2341548
latency average: 0.978 ms
tps = 3902.258427 (including connections establishing)
tps = 3902.821141 (excluding connections establishing)
```

```
postgres@test0:~$ dropdb bench1 --if-exists
postgres@test0:~$ createdb bench1
postgres@test0:~$ pgbench -i -s 600 bench1

postgres@test0:~$ pgbench -c 4 -j 2 -T 600 -S bench1
starting vacuum...end.
transaction type: SELECT only
scaling factor: 600
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 2288281
latency average: 0.996 ms
tps = 3813.521815 (including connections establishing)
tps = 3814.142184 (excluding connections establishing)
```

Execucions amb peticions de només escriptura al node *Master,* accedint a la memòria

```
postgres@tuco1:~$ dropdb bench1 --if-exists
postgres@tuco1:~$ createdb bench1
postgres@tuco1:~$ pgbench -i -s 70 bench1

postgres@tuco1:~$ pgbench -c 4 -j 2 -T 600 -N bench1
starting vacuum...end.
transaction type: Update only pgbench_accounts
scaling factor: 70
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 359896
latency average: 5.840 ms
tps = 599.648249 (including connections establishing)
tps = 600.001517 (excluding connections establishing)
```

```
postgres@tuco1:~$ dropdb bench1 --if-exists
postgres@tuco1:~$ createdb bench1
postgres@tuco1:~$ pgbench -i -s 70 bench1

postgres@tuco1:~$ pgbench -c 4 -j 2 -T 600 -N bench1
starting vacuum...end.
transaction type: Update only pgbench_accounts
scaling factor: 70
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 389587
latency average: 5.673 ms
tps = 649.138458 (including connections establishing)
tps = 649.526984 (excluding connections establishing)
```

```
postgres@tuco1:~$ dropdb bench1 --if-exists
postgres@tuco1:~$ createdb bench1
postgres@tuco1:~$ pgbench -i -s 70 bench1

postgres@tuco1:~$ pgbench -c 4 -j 2 -T 600 -N bench1
starting vacuum...end.
transaction type: Update only pgbench_accounts
scaling factor: 70
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 401548
latency average: 5.621 ms
tps = 669.105711 (including connections establishing)
tps = 669.497236 (excluding connections establishing)
```

```
postgres@tuco1:~$ dropdb bench1 --if-exists
postgres@tuco1:~$ createdb bench1
postgres@tuco1:~$ pgbench -i -s 70 bench1

postgres@tuco1:~$ pgbench -c 4 -j 2 -T 600 -N bench1
starting vacuum...end.
transaction type: Update only pgbench_accounts
scaling factor: 70
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 421554
latency average: 5.574 ms
tps = 702.478425 (including connections establishing)
tps = 702.841104 (excluding connections establishing)
```

```
postgres@tuco1:~$ dropdb bench1 --if-exists
postgres@tuco1:~$ createdb bench1
postgres@tuco1:~$ pgbench -i -s 70 bench1

postgres@tuco1:~$ pgbench -c 4 -j 2 -T 600 -N bench1
starting vacuum...end.
transaction type: Update only pgbench_accounts
scaling factor: 70
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 434814
latency average: 5.527 ms
tps = 724.501816 (including connections establishing)
tps = 724.856780 (excluding connections establishing)
```

Execucions amb peticions de només escriptura del servidor amb la instal·lació típica, accedint a la memòria

```
postgres@test0:~$ dropdb bench1 --if-exists
postgres@test0:~$ createdb bench1
postgres@test0:~$ pgbench -i -s 70 bench1

postgres@test0:~$ pgbench -c 4 -j 2 -T 600 -N bench1
starting vacuum...end.
transaction type: Update only pgbench_accounts
scaling factor: 70
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 433549
latency average: 5.536 ms
tps = 722.572297 (including connections establishing)
tps = 722.592286 (excluding connections establishing)
```

```
postgres@test0:~$ dropdb bench1 --if-exists
postgres@test0:~$ createdb bench1
postgres@test0:~$ pgbench -i -s 70 bench1

postgres@test0:~$ pgbench -c 4 -j 2 -T 600 -N bench1
starting vacuum...end.
transaction type: Update only pgbench_accounts
scaling factor: 70
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 413912
latency average: 5.798 ms
tps = 689.845103 (including connections establishing)
tps = 689.865726 (excluding connections establishing)
```

```
postgres@test0:~$ dropdb bench1 --if-exists
postgres@test0:~$ createdb bench1
postgres@test0:~$ pgbench -i -s 70 bench1

postgres@test0:~$ pgbench -c 4 -j 2 -T 600 -N bench1
starting vacuum...end.
transaction type: Update only pgbench_accounts
scaling factor: 70
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 441548
latency average: 5.510 ms
tps = 735.795847 (including connections establishing)
tps = 736.076553 (excluding connections establishing)
```

```
postgres@test0:~$ dropdb bench1 --if-exists
postgres@test0:~$ createdb bench1
postgres@test0:~$ pgbench -i -s 70 bench1

postgres@test0:~$ pgbench -c 4 -j 2 -T 600 -N bench1
starting vacuum...end.
transaction type: Update only pgbench_accounts
scaling factor: 70
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 429265
latency average: 5.595 ms
tps = 715.254813 (including connections establishing)
tps = 715.601529 (excluding connections establishing)
```

```
postgres@test0:~$ dropdb bench1 --if-exists
postgres@test0:~$ createdb bench1
postgres@test0:~$ pgbench -i -s 70 bench1

postgres@test0:~$ pgbench -c 4 -j 2 -T 600 -N bench1
starting vacuum...end.
transaction type: Update only pgbench_accounts
scaling factor: 70
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 444750
latency average: 5.396 ms
tps = 741.167923 (including connections establishing)
tps = 741.195573 (excluding connections establishing)
```

Execucions amb peticions de només escriptura al node *Master,* accedint al disc

```
postgres@tuco1:~$ dropdb bench1 --if-exists
postgres@tuco1:~$ createdb bench1
postgres@tuco1:~$ pgbench -i -s 600 bench1

postgres@tuco1:~$ pgbench -c 4 -j 2 -T 600 -N bench1
starting vacuum...end.
transaction type: Update only pgbench_accounts
scaling factor: 600
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 271273
latency average: 8.356 ms
tps = 452.212332 (including connections establishing)
tps = 452.264727 (excluding connections establishing)
```

```
postgres@tuco1:~$ dropdb bench1 --if-exists
postgres@tuco1:~$ createdb bench1
postgres@tuco1:~$ pgbench -i -s 600 bench1

postgres@tuco1:~$ pgbench -c 4 -j 2 -T 600 -N bench1
starting vacuum...end.
transaction type: Update only pgbench_accounts
scaling factor: 600
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 345188
latency average: 6.953 ms
tps = 575.285038 (including connections establishing)
tps = 575.304399 (excluding connections establishing)
```

```
postgres@tuco1:~$ dropdb bench1 --if-exists
postgres@tuco1:~$ createdb bench1
postgres@tuco1:~$ pgbench -i -s 600 bench1

postgres@tuco1:~$ pgbench -c 4 -j 2 -T 600 -N bench1
starting vacuum...end.
transaction type: Update only pgbench_accounts
scaling factor: 600
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 347815
latency average: 6.900 ms
tps = 579.683803 (including connections establishing)
tps = 579.701904 (excluding connections establishing)
```

```
postgres@tuco1:~$ dropdb bench1 --if-exists
postgres@tuco1:~$ createdb bench1
postgres@tuco1:~$ pgbench -i -s 600 bench1

postgres@tuco1:~$ pgbench -c 4 -j 2 -T 600 -N bench1
starting vacuum...end.
transaction type: Update only pgbench_accounts
scaling factor: 600
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 302137
latency average: 7.943 ms
tps = 503.530461 (including connections establishing)
tps = 503.547569 (excluding connections establishing)
```

```
postgres@tuco1:~$ dropdb bench1 --if-exists
postgres@tuco1:~$ createdb bench1
postgres@tuco1:~$ pgbench -i -s 600 bench1

postgres@tuco1:~$ pgbench -c 4 -j 2 -T 600 -N bench1
starting vacuum...end.
transaction type: Update only pgbench_accounts
scaling factor: 600
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 288215
latency average: 8.014 ms
tps = 480.011478 (including connections establishing)
tps = 480.079231 (excluding connections establishing)
```

Execucions amb peticions de només escriptura del servidor amb la instal·lació típica, accedint al disc

```
postgres@test0:~$ dropdb bench1 --if-exists
postgres@test0:~$ createdb bench1
postgres@test0:~$ pgbench -i -s 600 bench1

postgres@tuco0:~$ pgbench -c 4 -j 2 -T 600 -N bench1
starting vacuum...end.
transaction type: Update only pgbench_accounts
scaling factor: 600
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 362113
latency average: 6.628 ms
tps = 603.501901 (including connections establishing)
tps = 603.520365 (excluding connections establishing)
```

```
postgres@test0:~$ dropdb bench1 --if-exists
postgres@test0:~$ createdb bench1
postgres@test0:~$ pgbench -i -s 600 bench1

postgres@tuco0:~$ pgbench -c 4 -j 2 -T 600 -N bench1
starting vacuum...end.
transaction type: Update only pgbench_accounts
scaling factor: 600
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 366865
latency average: 6.607 ms
tps = 611.235854 (including connections establishing)
tps = 603.605488 (excluding connections establishing)
```

```
postgres@test0:~$ dropdb bench1 --if-exists
postgres@test0:~$ createdb bench1
postgres@test0:~$ pgbench -i -s 600 bench1

postgres@tuco0:~$ pgbench -c 4 -j 2 -T 600 -N bench1
starting vacuum...end.
transaction type: Update only pgbench_accounts
scaling factor: 600
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 381655
latency average: 6.288 ms
tps = 635.865504 (including connections establishing)
tps = 635.890650 (excluding connections establishing)
```

```
postgres@test0:~$ dropdb bench1 --if-exists
postgres@test0:~$ createdb bench1
postgres@test0:~$ pgbench -i -s 600 bench1

postgres@tuco0:~$ pgbench -c 4 -j 2 -T 600 -N bench1
starting vacuum...end.
transaction type: Update only pgbench_accounts
scaling factor: 600
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 419143
latency average: 5.726 ms
tps = 698.560271 (including connections establishing)
tps = 698.595996 (excluding connections establishing)
```

```
postgres@test0:~$ dropdb bench1 --if-exists
postgres@test0:~$ createdb bench1
postgres@test0:~$ pgbench -i -s 600 bench1

postgres@tuco0:~$ pgbench -c 4 -j 2 -T 600 -N bench1
starting vacuum...end.
transaction type: Update only pgbench_accounts
scaling factor: 600
query mode: simple
number of clients: 4
number of threads: 2
duration: 600 s
number of transactions actually processed: 388754
latency average: 6.430 ms
tps = 647.741795 (including connections establishing)
tps = 648.201139 (excluding connections establishing)
```